BENDERS DECOMPOSITION: AN INTEGER-PROGRAMMING EXTENSION WITH FURTHER COMPUTATIONAL ENHANCEMENTS

by

David A. Tarvin
© Copyright by David A. Tarvin, 2014
All Rights Reserved
A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Interdisciplinary).

Golden, Colorado
Date ____________________

Signed: ____________________
   David A. Tarvin

Signed: ____________________
   Dr. Alexandra M. Newman
   Thesis Advisor

Signed: ____________________
   Dr. R. Kevin Wood
   Thesis Advisor

Golden, Colorado
Date ____________________

Signed: ____________________
   Dr. Michael R. Walls
   Professor and Division Director
   Division of Economics and Business
ABSTRACT

We extend Benders decomposition in two ways. We begin by introducing a new integer Benders decomposition algorithm (IBDA) that solves pure integer programs (IPs). IBDA solves a mixed-integer relaxation of the IP by Benders decomposition and then conducts a type of local search to identify promising solutions to the original problem. IBDA’s key contributions are the local-search technique and a novel use of solution-elimination constraints. We prove IBDA’s correctness and demonstrate that the algorithm solves certain IPs faster than other available techniques. Slow Benders master-problem solution times can limit IBDA’s effectiveness, however. To ameliorate this problem, we therefore develop a “Benders decomposition algorithm using enumeration to solve master problems” (BDEA). BDEA stores objective-function values for all master-problem solutions, and then solves the subsequent master problem by comparing the incumbent value to the value of the most recent Benders cut for every feasible solution. Using enumeration, master-problem solution times remain constant even as the number of Benders cuts increases. We demonstrate BDEA’s performance using a stochastic capacitated facility-location problem. Computational tests show that BDEA can reduce average solution times by up to 74% compared to a standard BDA implementation.
# TABLE OF CONTENTS

ABSTRACT ......................................................... iii

LIST OF FIGURES ................................................. vii

LIST OF TABLES .................................................. ix

ACKNOWLEDGMENTS ............................................... 1

CHAPTER 1 GENERAL INTRODUCTION .............................. 2

CHAPTER 2 SOLVING PURE INTEGER PROGRAMS WITH BENDERS DECOMPOSITION ............................................. 5

2.1 Abstract .................................................... 5

2.2 Introduction .................................................. 5

2.3 Benders Decomposition for Mixed-Integer Programs ......... 7

2.3.1 Method Overview ........................................... 8

2.3.2 Benders Decomposition Algorithm (BDA) .................. 10

2.4 Benders Decomposition Applied to Integer Programs ........ 11

2.4.1 Integer Programs that Solve by Benders Decomposition Algorithm . . . 12

2.4.2 Integer Benders Decomposition Algorithm (IBDA) ........... 14

2.5 Set-Packing Problem ........................................... 18

2.5.1 Instances that Yield Totally Unimodular Subproblems .......... 19

2.5.2 Nonbipartite Maximum Matching .......................... 20

2.6 Computational Results ........................................ 21

2.6.1 Instances with a Bipartite-Graph Subproblem ............... 21
2.6.2 Instances with Consecutive 1’s Property in the Subproblem
Constraint Matrix ....................................................... 23

2.6.3 Practical Application with an Alternate Master-Problem Solution
Technique ................................................................. 26

2.7 Conclusions ............................................................ 28

CHAPTER 3 USING ENUMERATION TO SOLVE BINARY MASTER
PROBLEMS IN BENDERS DECOMPOSITION ...................... 30

3.1 Abstract ................................................................. 30

3.2 Introduction ............................................................. 30

3.3 Background ............................................................... 33

3.3.1 Binary Mixed Integer Program ........................................ 33

3.3.2 Benders Decomposition Algorithm (BDA) ........................... 34

3.4 Master Problem Solution by Enumeration ......................... 36

3.4.1 Benders Decomposition Algorithm Using Enumeration to Solve
Master Problems (BDEA) .................................................. 36

3.4.2 Knapsack Enumeration Algorithm (KEA) ........................... 38

3.4.3 Parallel Processing ..................................................... 39

3.4.4 Multicut Master Problem ............................................. 40

3.5 Computational Results .................................................. 40

3.5.1 Basic Results ......................................................... 41

3.5.2 Lazy Constraints ..................................................... 43

3.6 Conclusions ............................................................... 44

CHAPTER 4 SOLVING A CLASS OF STOCHASTIC MIXED-INTEGER
PROGRAMS BY BENDERS DECOMPOSITION WITH
ENUMERATED MASTER-PROBLEM SOLUTIONS .......... 45

4.1 Abstract ................................................................. 45
# LIST OF FIGURES

| Figure 2.1 | Benders decomposition algorithm for a mixed integer-program (BDA). | 10 |
| Figure 2.2 | Integer Benders decomposition algorithm (IBDA). | 16 |
| Figure 2.3 | Partitioning a complete graph into a largest possible bipartite graph, represented by $D$, and a non-bipartite graph, represented by $B$. | 22 |
| Figure 2.4 | $D$-matrix with consecutive 1’s property, where the maximum number of consecutive 1’s $\tau = 3$. The left side of $D$ contains all possible columns with two consecutive 1’s, while the right side contains all possible columns with three consecutive 1’s. | 24 |
| Figure 3.1 | Benders decomposition algorithm (BDA). | 35 |
| Figure 3.2 | BDA using enumeration to solve master problems (BDEA). | 37 |
| Figure 3.3 | Knapsack enumeration algorithm, modified from Horowitz and Sahni (1978). | 39 |
| Figure 3.4 | Comparing methods for solving Benders master problems. (a) The plot depicts the total master problem solution time (in seconds) for a specified number of Benders iterations. Note that the solution times are shown on a logarithmic scale. (b) This plot truncates the vertical scale to emphasize that total run time for explicit enumeration is essentially linear in the number of master problems solved, $K$. (The initial enumeration of $\bar{\mathcal{X}}$, the set of maximal solutions, creates an offset at $K = 0$ on the logarithmic scale, but that offset is clearly negligible here.) | 42 |
| Figure 4.1 | Single-cut Benders decomposition algorithm for $2SSP(\mathcal{X}, \mathcal{Y}, \mathcal{S})$ (SCA). | 50 |
| Figure 4.2 | Multicut Benders decomposition algorithm for $2SSP(\mathcal{X}, \mathcal{Y}, \mathcal{S})$ (MCA). | 51 |
| Figure 4.3 | Single-cut BDA using enumeration to solve master problems for $2SSP(\mathcal{X}, \mathcal{Y}, \mathcal{S})$ (SCEA). | 53 |
| Figure 4.4 | Multicut BDA using enumeration to solve master problems for $2SSP(\mathcal{X}, \mathcal{Y}, \mathcal{S})$ (MCEA). | 54 |
Figure 4.5  Binary-addition method for calculating $\eta(x)$. Assumes that $|S| = 2^n$ for some integer $n > 0$. ................................. 56
LIST OF TABLES

Table 2.1  Comparison of IBDA and B&B run times for a maximum matching problem on a 1000-node complete graph. For each combination of relative optimality tolerance and distribution of observations, we solve ten randomly generated instances of \texttt{PACK}(\mathcal{X}, \mathcal{Y}) and report the average CPLEX time in seconds. ................................. 23

Table 2.2  Comparison of IBDA and B&B run times for set-packing problems whose D-matrix has the COP. For each combination of optimality tolerance ($\varepsilon$), $p$, and $\tau$, we solve five randomly generated \texttt{PACK}(\mathcal{X}, \mathcal{Y}) instances and report the average CPLEX time in seconds. ................................. 25

Table 2.3  Statistics for IBDA and B&B solving ten random instances of the stochastic facility location problem $S(I, J, b, \beta, 50)$ using relative optimality tolerance $\varepsilon = 0.01$. ................................. 27

Table 3.1  Solution times for B&B solving the final 30 (of 1400) master problems while letting either the oldest or the loosest cuts be lazy. ................................. 44

Table 4.1  Statistics to compare B&B and various decomposition algorithms for solving ten random model instances of $P(20, 50, 10, \beta, |S|)$, using $N = 1, 4$, and 8 processors. ................................. 59

Table 4.2  Statistics to compare B&B and various decomposition algorithms for solving ten random model instances of $P(20, 50, 10, \beta, |S|)$, using $\varepsilon_{BD} = 0.01$. ................................. 61

Table A.1  Stochastic capacitated facility-location description from Sanchez and Wood (2006). ................................. 78

Table A.2  Parameters for stochastic, capacitated facility-location test problems. ................................. 80
ACKNOWLEDGMENTS

To my advisors, Professors Alexandra Newman and Kevin Wood, thank you for your guidance, mentorship, and patience. To my committee members, Professors Mark Coffey, Dinesh Mehta, and Cameron Turner, thank you for the time and energy that you have each committed to this process. To Professor Javier Salmerón, thank you for the opportunity to collaborate with you.

I would not have had this opportunity without the motivation and support of a tremendous number of Army officers and civilians. With my apologies to anyone I may have forgotten, thank you to: Lieutenant General Joseph Martz, Brigadier General Cedric Wins, Colonel Steve Grimes, Colonel Mark Lukens, Lieutenant Colonel Kaye McKinzie, and, most of all, Miss Karen Miller.

Most importantly, thank you to Kyle and Liz. Without the support that you have provided and the sacrifices that you have made, this would not have been possible. I am very fortunate to be a part of your lives.
CHAPTER 1
GENERAL INTRODUCTION

The purpose of this thesis is two-fold: first, to extend Benders decomposition to solve pure integer programs, and second, to develop computational enhancements that improve Benders decomposition’s efficiency.

This document contains chapters for each of three journal-style papers that describe a portion of our research. David A. Tarvin is the primary researcher and author for all chapters, which are independent, except that the Appendix applies to both Chapter 2 and Chapter 4. Throughout, we assume that the reader is familiar with basic integer programming applications and solution techniques at the level of Chapters I.1-I.2 and II.3-II.5 in Nemhauser and Wolsey (1999).

Chapter 2 presents an integer Benders decomposition algorithm (IBDA) that extends Benders decomposition to linear integer programs (IPs). The basic integer program has this form:

\[ \text{IP}(\mathcal{X}, \mathcal{Y}) \quad \min \{ cx + fy \mid Ax \geq b, Bx + Dy \geq d, x \in \{0, 1\}^{n_x}, y \in \{0, 1\}^{n_y} \}, \quad (1.1) \]

where \( A, B, \) and \( D \) are dimensioned \( m_1 \times n_x, m_2 \times n_x, \) and \( m_2 \times n_y, \) respectively, and all vectors conform. While we specialize to a binary IP for simplicity, our methods extend, in a standard fashion, to general IPs with bounded variables.

IBDA relaxes variables \( y \) to be continuous on the range \([0, 1]\), which yields the following mixed-integer program (MIP):

\[ \text{MIP}(\mathcal{X}, \mathcal{Y}) \quad \min \{ cx + fy \mid Ax \geq b, Bx + Dy \geq d, x \in \{0, 1\}^{n_x}, 0 \leq y \leq 1 \}, \quad (1.2) \]

where \( y \) denotes the relaxed version of \( y \).
IBDA then solves $\text{MIP}(X, Y)$ by Benders decomposition for an optimal solution $(x^*, y^*)$. If $y^*$ is integral, then $(x^*, y^*)$ solves the IP. Otherwise, IBDA uses a “local-search” technique to identify promising integer solutions, applies solution-elimination constraints to guarantee convergence, and resumes Benders decomposition iterations.

After developing IBDA, we test its efficiency by solving set-packing problems with special structure. IBDA does solve certain instances of the set-packing problem faster than a standard branch-and-bound (B&B) algorithm. We find that many test-problem instances yield relatively slow Benders decomposition solution times, however. These difficulties prompt the development, in Chapters 3 and 4, of special computational techniques to speed up the solution of the Benders master-problems, which we use to solve integer programs in Chapter 2.

Chapter 3 addresses one of the key computational difficulties for the Benders decomposition algorithm: the need to solve the mixed-integer master problem repeatedly. Each Benders iteration introduces a new master-problem constraint, a “Benders cut,” which increases the master problem’s size and worst-case solution time (Nemhauser and Wolsey, 1999, p. 125). We consider a MIP that takes the form in (1.2), but with no upper bounds on $y$, and introduce a “Benders decomposition algorithm that uses enumeration to solve the master problem” (BDEA).

BDEA stores the objective-function value for every feasible master-problem solution, thus avoiding the need to solve master problems “from scratch” during each iteration. Instead, BDEA solves each iteration’s master problem by enumerating all feasible master-problem solutions and, for each, (i) computing the lower bound that the new Benders cut provides; (ii) comparing the new lower bound to the previous lower bound, that is, to the stored objective-function value; and (iii) saving the greater of the two bounds as the new objective-function value. We evaluate BDEA’s potential by solving a sequence of master problems generated while applying a variant of Benders to a network-interdiction problem. Improvements possible with parallel processing are also investigated.
Chapter 4 investigates BDEA performance by comparing MIP solution times obtained (i) using BDEA, (ii) using a typical Benders decomposition implementation, and (iii) using B&B to directly solve the monolith. We specialize to a two-stage stochastic mixed-integer program (2SSP) with binary first-stage variables and a tractable equivalent deterministic formulation:

\[
\begin{align*}
2SSP(\mathcal{X}, \mathcal{Y}, \mathcal{S}) \quad z^* &= \min \mathbf{c} \mathbf{x} + \sum_{s \in \mathcal{S}} p^s \mathbf{f} \mathbf{y}^s \\
\text{s.t.} \quad A \mathbf{x} &\geq \mathbf{b} \\
B \mathbf{x} + D \mathbf{y}^s &\geq \mathbf{d}^s \quad \forall \ s \in \mathcal{S}, \\
\mathbf{x} &\in \{0, 1\}^n \\
\mathbf{y}^s &\geq 0 \quad \forall \ s \in \mathcal{S},
\end{align*}
\]  

where \(\mathcal{S}\) is a set of scenarios, \(p^s\) is the probability that scenario \(s \in \mathcal{S}\) occurs, and here \(\mathbf{y}^s\) represents a vector of continuous variables; superscripts denote scenarios. During each iteration, a multicut BDEA generates \(|\mathcal{S}|\) Benders master-problem cuts, one per scenario subproblem. The original single-cut variant, in effect, combines these \(|\mathcal{S}|\) cuts into one master-problem constraint.

Our research contributions are: (i) we extend Benders decomposition to solve pure IPs; (ii) we develop an enumerative method that can reduce Benders master-problem solution times by an order of magnitude; and (iii) we show that BDEA can reduce 2SSP solution times by up to 74% compared to a standard BDA implementation.
CHAPTER 2
SOLVING PURE INTEGER PROGRAMS WITH BENDERS DECOMPOSITION

Modified from an article written for Qualifying Examination II
D. Antony Tarvin

2.1 Abstract

We lay new groundwork for extending Benders decomposition to linear integer programs (IPs). Initially specializing to a set-packing problem with decision-variable vector arbitrarily partitioned into \((x, y)\), we relax the IP to yield a mixed-integer program (MIP) that we solve for \((x^*, y^*)\) with Benders decomposition, where \(y\) represents the continuous relaxation of \(y\). If \(y^*\) is not integral, we then: (i) solve two related IPs to identify promising integer solutions, (ii) ensure finite convergence by introducing solution-elimination constraints (SECs) to the Benders master problem and to the related IPs, and (iii) resume Benders decomposition iterations. Our key contributions are a means to identify promising integer solutions from a given MIP solution and a novel application of SECs. Computational results identify promising partitioning strategies for IPs and demonstrate that our method can solve certain IPs faster—more than 90% faster in some tests—than does a standard branch-and-bound algorithm.

2.2 Introduction

Dantzig-Wolfe decomposition was originally intended to solve linear programs (LPs) (Dantzig and Wolfe, 1961) and has since been extended to integer programs (IPs) through branch-and-price (e.g., Barnhart et al., 1998; Singh et al., 2009). Benders decomposition solves LPs and mixed-integer programs (MIPs) (Benders, 1962), but an extension to general IPs has been elusive. This chapter makes that extension.
While researchers have certainly examined Benders decomposition as it relates to solving pure integer programs (as discussed below), Ralphs and Galati (2005) make no mention of Benders decomposition algorithm (BDA) in their review of IP decomposition. Vanderbeck and Wolsey (2010) introduce a Benders-like method for IPs, although without demonstrating its utility; we discuss this method in more detail later. Multiple authors have adapted Benders decomposition to solve two-stage stochastic programs (2SSPs) with purely integer variables, however. For example, Carøe and Tind (1998) use “Gomory’s Fractional Cutting Plane Algorithm” to generate cutting planes that solve the second-stage scenario subproblems, although the technique tends to lead to computationally intractable first-stage problems (that is, master problems). Sherali and Fraticelli (2002) also use cutting planes to solve subproblems; these authors introduce cuts that can be recomputed and reused for subsequent subproblems. Sen and Higle (2005) employ disjunctive programming (Balas, 1985) to generate sequential convexifications of the integer master problem and of the integer subproblems. In Section 2.4, we examine the final two methods in more detail. Alternatively, Lulli and Sen (2004) and Silva and Wood (2006) each employ a branch-and-price algorithm (B&P) to solve 2SSPs. B&P requires a reformulation of the subproblems, however, and we will limit this chapter to techniques that avoid reformulation.

We propose a new integer Benders decomposition algorithm (IBDA) that partitions the integer decision-variable vector into \((x, y)\), temporarily relaxes \(y\) to continuous \(\bar{y}\), and then iteratively solves the resulting MIP with BDA, generates promising integer solutions, eliminates those solutions from future consideration, and resumes BDA iterations. Our integer Benders method differs significantly from those of other researchers in that we obtain an integer optimal solution while both maintaining a tractable master problem and avoiding complicated subproblem cuts. We present IBDA for the special case of binary IPs and, when applying Benders decomposition to the relaxed MIP, we assume subproblem feasibility given any feasible master problem solution. The assumption is valid for the aircraft-routing problem of Richardson (1976) and the capacitated facility-location problem of Van Roy (1986),
and is made in numerous papers (e.g., Rockafellar and Wets, 1976; Magnanti and Wong, 1984; Verweij et al., 2003).

Of course, if an IP yields integer solutions without enforcing integer restrictions on the subproblem variables, then a standard BDA will solve the IP. Therefore, we expect that strategies for partitioning variables into the temporarily relaxed \( y \) and the not-relaxed \( x \) may be important, and we explore two schemes that guarantee integer subproblem solutions for IPs with particular forms.

For general IPs, IBDA augments Benders decomposition in two ways. First, we introduce two related IPs that support a “local search” for optimal IP solutions, given the solution to the mixed-integer relaxation. Second, we employ solution-elimination constraints (SECs) to ensure finite convergence of the algorithm. We introduce SECs to the master problem, as in Brown et al. (2009). However, we also extend these authors’ work by incorporating SECs into our local-search IPs.

Computational tests evaluate set-packing problems and compare the efficiency of IBDA to that of a standard linear-programming-based branch-and-bound algorithm. We also examine a facility-location problem from Sanchez and Wood (2006).

The rest of this chapter is organized as follows. Section 2.3 introduces a standard BDA for solving a MIP. Section 2.4 proposes the IBDA. Section 2.5 presents the set packing problem as a practical example. Section 2.6 contains IBDA testing results and discussion. Section 2.7 provides conclusions and recommendations for future research.

### 2.3 Benders Decomposition for Mixed-Integer Programs

The theory here extends easily to IPs with bounded variables but, for notational simplicity, we consider only binary IPs in the following standard form:

\[
\text{IP}(\mathcal{X}, \mathcal{Y}) \quad z^*_x = \min_{x \in \mathcal{X}, y \in \mathcal{Y}} cx + fy \\
\text{s.t.} \quad Ax \geq b \\
Bx + Dy \geq d,
\]

(2.1)

(2.2)

(2.3)
where $\mathcal{X} \equiv \{x \in \{0,1\}^{n_x}\}$; $\mathcal{Y} \equiv \{y \in \{0,1\}^{n_y}\}$; $A$, $B$, and $D$ are dimensioned $m_1 \times n_x$, $m_2 \times n_x$, and $m_2 \times n_y$, respectively; and the vectors $b$, $c$, $d$, $f$, $x$, and $y$ conform. While we can solve $\text{IP} (\mathcal{X}, \mathcal{Y})$ directly by using branch-and-bound, Benders decomposition would normally be applied to a model with the structure of $\text{IP} (\mathcal{X}, \mathcal{Y})$ only if $y$ is continuous.

Benders decomposition is the foundation of our algorithm for solving $\text{IP} (\mathcal{X}, \mathcal{Y})$. Thus, to provide a reference point and to introduce notation, we present a standard BDA to solve a MIP. For illustration here, and as part of the IBDA introduced in Section 2.4.2, we relax $\text{IP} (\mathcal{X}, \mathcal{Y})$ to create $\text{MIP} (\mathcal{X}, \mathcal{Y})$, with $\overline{y}$ denoting the relaxed version of $y$:

$$\text{MIP} (\mathcal{X}, \mathcal{Y}) \quad z^*_x = \min_{x \in \mathcal{X}, \overline{y} \in \mathcal{Y}} cx + fy$$

s.t. $Ax \geq b$ \hspace{1cm} (2.5)

$Bx + Dy \geq d$, \hspace{1cm} (2.6)

where $\mathcal{Y} \equiv \{y \in \mathbb{R}^{n_y} \mid y \leq 1\}$, and $\mathbb{R}^k_+$ denotes the $k$-dimensional, non-negative real space.

We assume the following for $\text{MIP} (\mathcal{X}, \mathcal{Y})$:

**Assumption 2.1** Let $\text{MIP} (\hat{x}, \mathcal{Y})$ denote $\text{MIP} (\mathcal{X}, \mathcal{Y})$ with $x \equiv \hat{x}$. $\text{MIP} (\hat{x}, \mathcal{Y})$ is feasible for any $\hat{x} \in \{0,1\}^{n_x}$ that satisfies constraints (2.5). \hfill \blacksquare

Assumption 2.1 corresponds to the assumption of “relatively complete recourse,” which is commonly made in the stochastic-programming literature (e.g., Ahmed et al., 2004; Sen and Higle, 2005). We make the assumption here for expository simplicity and justify it directly below.

### 2.3.1 Method Overview

If $x = \hat{x} \in \mathcal{X}$, then $\text{MIP} (\mathcal{X}, \mathcal{Y})$ yields the following subproblem:

$$\text{SUB} (\hat{x}, \mathcal{Y}) \quad z^*(\hat{x}) = c\hat{x} + \min_{\overline{y} \in \mathcal{Y}} fy$$

s.t. $D\overline{y} \geq d - B\hat{x}$ \hspace{1cm} \overline{\alpha(\hat{x})}$ \hspace{1cm} (2.7)

$-1\overline{y} \geq -1$ \hspace{1cm} \overline{\beta(\hat{x})}$, \hspace{1cm} (2.8)

where $\overline{\alpha(\hat{x})}$ and $\overline{\beta(\hat{x})}$ are the $\hat{x}$-dependent Benders cuts.
where vectors of optimal dual-variable values are shown in brackets next to the relevant constraints. For simplicity, let \( \hat{\alpha} = \alpha(\hat{x}) \) and let \( \hat{\beta} = \beta(\hat{x}) \).

By Assumption 2.1, \( \text{SUB}(\hat{x}, \mathcal{Y}) \) has an optimal solution for each \( \hat{x} \) that satisfies the master-problem constraints. While convenient, Assumption 2.1 is not limiting. For problems in which \( \text{SUB}(\hat{x}, \mathcal{Y}) \) does not necessarily have a feasible solution, BDA can add “feasibility cuts” to the master problem, as necessary (Benders, 1962). Alternatively, BDA can solve a reformulated subproblem that permits violations of constraints (2.8) and then penalizes these violations in the objective function (2.7) (Yuen et al., 2006). For this second method, we change Assumption 2.1 to read “… with \( x \equiv \hat{x} \). We assume that the subproblem has an optimal solution for any \( \hat{x} \in \{0, 1\}^n \) that satisfies constraints (2.5).”

In its standard incarnation, BDA reformulates \( \text{MIP}(\mathcal{X}, \mathcal{Y}) \) into an equivalent master problem, \( \text{MP}(\mathcal{X}, \mathcal{A}) \), where \( \mathcal{A} \) is the set of all extreme-point vectors for \( \{(\alpha, \beta) | \alpha D - \beta \leq f, \alpha \geq 0, \beta \geq 0\} \).

\[
\text{MP}(\mathcal{X}, \mathcal{A}) \quad z^* = \min_{x \in \mathcal{X}, \eta} \eta \\
\text{s.t.} \quad Ax \geq b \quad (2.10) \\
\eta \geq cx + \hat{\alpha}(d - Bx) - \hat{\beta}1 \quad \forall (\hat{\alpha}, \hat{\beta}) \in \mathcal{A}. \quad (2.11)
\]

Constraints (2.12) reflect the contribution to \( z^*_x \) from \( \mathcal{Y} \) and associated constraints.

Rather than exhaustively calculating \( \mathcal{A} \) and then solving \( \text{MP}(\mathcal{X}, \mathcal{A}) \) directly, BDA begins with a relaxed master problem, \( \text{MP}(\mathcal{X}, \hat{\mathcal{A}}) \), where \( \hat{\mathcal{A}} \) is a subset of \( \mathcal{A} \). The algorithm iteratively (i) solves \( \text{MP}(\mathcal{X}, \hat{\mathcal{A}}) \) for \( \hat{x} \), (ii) solves \( \text{SUB}(\hat{x}, \mathcal{Y}) \) for primal solution \( (\hat{x}, \hat{\mathcal{Y}}(\hat{x})) \) and associated dual solution \( (\hat{\alpha}, \hat{\beta}) \), (iii) adds \( (\hat{\alpha}, \hat{\beta}) \) to \( \hat{\mathcal{A}} \), and (iv) solves the new relaxed master problem. The algorithm terminates once \( \hat{\mathcal{A}} \) suffices for proving near-optimality of some previously discovered \( (\hat{x}, \hat{\mathcal{Y}}(\hat{x})) \) (Benders, 1962). (McDaniel and Devine 1977 present a similar description for the case with general integer variables.)
2.3.2 Benders Decomposition Algorithm (BDA)

To simplify the description of the integer Benders decomposition algorithm, we present BDA as a formal algorithm in Figure 2.1.

The correctness of BDA is evident from the following observations:

1. By Assumption 2.1, the subproblem is feasible and provides an optimal solution for any value of \( \hat{x} \). Thus, \( z^*(\hat{x}) \) must provide an upper bound on the objective function value of MIP(\( \mathcal{X}, \mathcal{Y} \)).

2. The master problem is a relaxation of MIP(\( \mathcal{X}, \mathcal{Y} \)). Therefore, \( z^*_K \) provides a lower bound on the MIP(\( \mathcal{X}, \mathcal{Y} \)) objective function value.

3. The final solution, \( (x^*, y^*) \), (i) is feasible for MIP(\( \mathcal{X}, \mathcal{Y} \)), (ii) provides an upper bound of \( \bar{z} \) on the MIP(\( \mathcal{X}, \mathcal{Y} \)) objective function value, and (iii) has an objective function

Figure 2.1: Benders decomposition algorithm for a mixed integer-program (BDA).

<table>
<thead>
<tr>
<th>Input</th>
<th>An instance of MIP(( \mathcal{X}, \mathcal{Y} )) and allowable optimality gap ( \varepsilon \geq 0 ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>An ( \varepsilon )-optimal solution to MIP(( \mathcal{X}, \mathcal{Y} )).</td>
</tr>
<tr>
<td>Note</td>
<td>( \bar{z} ) and ( \bar{z} ) are lower and upper bounds on ( z^*_{xy} ), and ( \hat{A} ) is a set of dual extreme-point vectors.</td>
</tr>
</tbody>
</table>

**Initialize**

1. Decompose MIP(\( \mathcal{X}, \mathcal{Y} \)) to MP(\( \mathcal{X}, \hat{A} \)) and SUB(\( \hat{x}, \mathcal{Y} \));
2. \( K \leftarrow 0; \bar{z} \leftarrow -\infty; \bar{z} \leftarrow \infty; \hat{A}_0 \leftarrow \emptyset; \hat{x} \leftarrow 0 \) (or any solution \( x \in \mathcal{X} \) that satisfies (2.5));

**Subproblem**

3. \( K \leftarrow K + 1 \);
4. Solve SUB(\( \hat{x}, \mathcal{Y} \)) for \( \hat{y} \), dual vector \( (\hat{\alpha}, \hat{\beta})_K \), and objective value \( z^*(\hat{x}) \);
5. \( \hat{A}_K \leftarrow \hat{A}_{K-1} \cup \{(\alpha, \beta)_K\} \);
6. if \( ( z^*(\hat{x}) < \bar{z} ) \) \{ \( \bar{z} \leftarrow z^*(\hat{x}); (x^*, y^*) \leftarrow (\hat{x}, \hat{y}) \) \} /* Update incumbent. */
7. if \( ( \bar{z} - \bar{z} \leq \varepsilon \bar{z} ) \) \{ Go to Step 11; \} /* We assume \( \bar{z}, \bar{z} \geq 0 \). */

/* Master Problem */

8. Solve MP(\( \mathcal{X}, \hat{A}_K \)) for \( \hat{x} \) and objective value \( z^*_K; \bar{z} \leftarrow z^*_K \); 
9. if \( ( \bar{z} - \bar{z} \leq \varepsilon \bar{z} ) \) \{ Go to Step 11; \}
10. Go to Step 3;

**Print Solution**

11. Print ("\( \varepsilon \)-optimal solution to MIP(\( \mathcal{X}, \mathcal{Y} \)) is," \( (\bar{z}, x^*, y^*) \)); Stop.
value that is within $\varepsilon$ of the $\text{MIP}(\mathcal{X}, \mathcal{Y})$ lower bound provided by the relaxed master problem $\text{MP}(\mathcal{X}, \hat{A})$. Since each BDA iteration generates a dual extreme-point vector that defines one constraint for $\text{MIP}(\mathcal{X}, \mathcal{Y})$, the algorithm terminates after a finite number of iterations with a solution to $\text{MIP}(\mathcal{X}, \mathcal{Y})$. 

2.4 Benders Decomposition Applied to Integer Programs

For $\hat{x} \in \mathcal{X}$, let $\mathcal{Y}(\hat{x})$ denote the feasible region for $\text{SUB}(\hat{x}, \mathcal{Y})$. Relaxing $y$ and applying BDA is guaranteed to solve $\text{IP}(\mathcal{X}, \mathcal{Y})$ only if every extreme point in $\mathcal{Y}(\hat{x})$ is integer for every feasible $\hat{x}$. Otherwise, we must extend Benders decomposition to solve the IP. Vanderbeck and Wolsey (2010) show how to do this for $\text{IP}(\mathcal{X}, \mathcal{Y})$ with binary $x$ and integer $y$. After solving $\text{MP}(\mathcal{X}, \hat{A})$ for $\hat{x}$, these authors solve an integer $\text{SUB}(\hat{x}, \mathcal{Y})$. If the subproblem is feasible, the authors generate an optimality cut for $\text{MP}(\mathcal{X}, \hat{A})$; otherwise, they generate a feasibility cut. For the method to avoid complete enumeration of the $n_x$-dimensional, binary space, the authors note the importance of strengthening the Benders cuts to eliminate more than just a single $\hat{x}$. To further limit enumeration, Vanderbeck and Wolsey focus on models with feasibility subproblems, that is, subproblems with $f = 0$, which enable the determination of subproblem solutions via constraint programming. (For example, see Wallace 1996 and Hooker 2002.)

Stochastic programmers commonly employ variants of Benders decomposition for sampled models and for models with a tractable deterministic equivalent, that is, with a relatively small number of subproblem realizations (e.g., Higle and Sen, 1999; Escudero et al., 2007; Sen et al., 2009). Multiple authors even extend Benders decomposition to 2SSPs with both integer first- and second-stage variables. For example, Laporte and Louveaux (1998) describe an “integer L-shaped method,” which relaxes integrality constraints, approximates the subproblem objective function value to obtain a lower bound, and then applies a branch-and-bound technique to produce an integer solution. If the lower bound is weak, however, the algorithm is prone to excessive enumeration (e.g., Laporte et al., 2002; Rei et al., 2007).
Sherali and Fraticelli (2002) solve 2SSPs with both integer first- and second-stage variables using a “Benders decomposition algorithm for discrete subproblems.” These authors solve the subproblem for an integer-optimal solution using either lift-and-project or partial convexification cuts. (See Balas et al. 1993 and Sherali et al. 2005.) Sherali and Fraticelli define their cuts as functions of $x$, so that they can be recalculated and re-used in subsequent iterations.

In their “disjunctive decomposition algorithm for large-scale stochastic MIPs,” Sen and Higle (2005) generate convex approximations of the master problem and of every scenario subproblem. During each iteration, their algorithm solves the approximated master problem, uses that solution to refine the scenario-subproblem approximations, solves the subproblems, and then generates a master-problem optimality cut from the subproblem solutions.

Rather than employing branch-and-cut or a convexification method to obtain an integer solution to the subproblem, we provide an algorithm that temporarily relaxes the IP to a MIP, which can then be solved with BDA. Our key developments are (i) a method for identifying a solution to the IP, given a solution to the MIP, and (ii) a novel application of SECs.

2.4.1 Integer Programs that Solve by Benders Decomposition Algorithm

Although meant to solve MIPs, BDA will actually solve some IPs (Sherali and Fraticelli, 2002). Proposition 2.1 formalizes this result, which is obvious and is stated without proof.

**Proposition 2.1** If MIP$(\mathcal{X}, \mathcal{Y})$ yields an integer optimal extreme-point solution for every $\hat{x} \in \mathcal{X}$ satisfying constraints (2.5), then an optimal solution $(x^*, y^*)$ to MIP$(\mathcal{X}, \mathcal{Y})$ solves IP$(\mathcal{X}, \mathcal{Y})$.

Certain problem structures guarantee that the MIP solution is necessarily integer, including a subset of problems with suproblem polytopes that are totally dual integral (TDI).

**Definition 2.1** (Giles and Pulleyblank, 1979) Let $Ax \geq b$ be a linear system with $A$ and $b$ rational. We say that this linear system is totally dual integral if and only if for any integer $x$. 

12
valued \( c \) such that the linear program

\[
\text{minimize } \{ cx | Ax \geq b \}
\]

has an optimal solution, the corresponding dual linear program has an integer optimal solution.

If \( Y(\hat{x}) \equiv \{ y \in \mathbb{R}^n_+ | Dy \geq d - B\hat{x}, y \leq 1 \} \) is a TDI subproblem polytope with rational \( D \) and integer \( d - B\hat{x} \), then \( Y(\hat{x}) \) is an integer polytope (Giles and Pulleyblank, 1979). For this case of an integer subproblem polytope, we extend Proposition 2.1:

**Proposition 2.2** If \( D \) is rational, and for all values of \( \hat{x} \in \mathcal{X} \) satisfying constraints (2.5) (i) \( Y(\hat{x}) \) is a TDI subproblem polytope, and (ii) \( d - B\hat{x} \) is integer, then Proposition 2.1 holds.

**Proof.** The polytope is TDI and \( d - B\hat{x} \) is integer. Therefore, every vertex of the polytope is integer-valued. By the fundamental theorem of linear programming, any linear program with an optimal solution contains at least one optimal extreme-point solution and Proposition 2.1 holds.

In certain cases, it may be possible to partition a MIP such that the resulting \( D \)-matrix ensures that Proposition 2.2 applies. One commonly studied subset of TDI polytopes having the form of \( Y(\hat{x}) \) is defined when \( D \) is totally unimodular (TU).

**Definition 2.2** (Tamir, 1976) \( D \) is a totally unimodular matrix if every square submatrix has determinant +1, −1, or 0.

For example, suppose that for every \( \hat{x} \in \mathcal{X} \) satisfying constraints (2.5), the corresponding subproblem is (i) a maximum cardinality matching in a bipartite graph (Edmonds, 1965), (ii) a network-design hub-node assignment problem (O’Kelly and Lao, 1991), or (iii) a network flow problem where \( d - B\hat{x} \) defines integer supplies and demands (Ahuja et al., 1993,
p. 449). Then $D$ is TU, Proposition 2.2 holds, and BDA will solve the IP provided the particular partitioning is used. We reserve the discussion of partitioning strategies for Section 2.5 and next present a general solution algorithm.

### 2.4.2 Integer Benders Decomposition Algorithm (IBDA)

When Proposition 2.1 does not apply, we want to augment BDA so that it solves $\text{IP}(\mathcal{X}, \mathcal{Y})$. To do so, we first introduce two related integer programs: (i) let $\text{IP}(\hat{x}, \mathcal{Y})$ be the integer program $\text{IP}(\mathcal{X}, \mathcal{Y})$, with $x \equiv \hat{x}$, and (ii) let $\text{IP}(\mathcal{X}, \hat{y})$ be the integer program $\text{IP}(\mathcal{X}, \mathcal{Y})$, with $y \equiv \hat{y}$.

After solving $\text{MIP}(\mathcal{X}, \mathcal{Y})$ by BDA for solution $(x^*, y^*)$, and solving the related integer programs $\text{IP}(\hat{x}, \mathcal{Y})$ and $\text{IP}(\mathcal{X}, \hat{y})$, IBDA may need to resume BDA iterations. To ensure that BDA does not repeatedly yield the same $(x^*, y^*)$, we introduce SECs that make a single integer vector $\hat{x}$ infeasible. We extend this concept by also using SECs to eliminate solutions $\hat{y}$. Nemhauser and Trick (1998) employ SECs (without naming them) to eliminate particular binary solutions in a scheduling model. These authors solve for a binary solution vector $x$, which must contain exactly $n$ non-zero elements. As part of their iterative method, the authors introduce an SEC to eliminate a particular binary solution vector, $\hat{x}$. Their SEC is:

$$\sum_{j|\hat{x}_j = 1} x_j \leq n - 1. \quad (2.13)$$

Since the more general $\text{IP}(\mathcal{X}, \mathcal{Y})$ has no restriction on the number of non-zero elements, we cannot simply change the right-hand side of equation (2.13) and use it as an SEC for IBDA. Israeli and Wood (2002) introduce “supervalid inequalities,” a generalization of SECs, that typically eliminate $\hat{x}$ and may eliminate other integer solutions as well. In their interdiction model, Brown et al. (2009) apply a more robust SEC formulation than that of Nemhauser and Trick. These authors solve for a binary solution vector $x$, which has no restriction on the number of non-zero elements. Their SEC, which we implement in IBDA, takes the form:
\[
\sum_{j: \hat{x}_j = 0} x_j + \sum_{j: \hat{x}_j = 1} (1 - x_j) \geq 1
\] (2.14)

and prevents the solution \( \hat{x} \) from being repeated, while permitting all others. (Nemhauser and Wolsey 1999, p. 126 note that an IP with general integer variables can be converted to
a binary IP, which would permit the application of the previous SEC.)

For \( \text{IP}(\mathcal{X}, \mathcal{Y}) \), let \( \hat{x} \) be a solution vector that we wish to eliminate from both \( \text{MP}(\mathcal{X}, \hat{A}) \) and \( \text{IP}(\mathcal{X}, \hat{y}) \). A constraint of the form in (2.14) makes \( \hat{x} \) infeasible in \( \text{MP}(\mathcal{X}, \hat{A}) \), and makes \( (\hat{x}, \hat{y}) \) infeasible in \( \text{IP}(\mathcal{X}, \hat{y}) \) for any \( \hat{y} \in \mathcal{Y} \), all without restricting any other \( x \in \mathcal{X} \). If \( \mathcal{X}^- \) is a set of \( x \) vectors to eliminate, then for each \( \hat{x} \in \mathcal{X}^- \) we add an SEC to both \( \text{MP}(\mathcal{X}, \hat{A}) \) and \( \text{IP}(\mathcal{X}, \hat{y}) \). We represent this set of constraints in shorthand notation as

\[
x \notin \mathcal{X}^-.
\] (2.15)

So, equations (2.10)-(2.12) plus (2.15) define \( \text{MP}(\hat{X}, \hat{A}) \), where \( \hat{X} \equiv \mathcal{X} \setminus \mathcal{X}^- \); \( \text{IP}(\hat{X}, \hat{y}) \) denotes \( \text{IP}(\mathcal{X}, \hat{y}) \) plus (2.15). Similarly, if \( \mathcal{Y}^- \) is a set of \( y \) vectors to eliminate, then we add the set of SECs

\[
y \notin \mathcal{Y}^-
\] (2.16)

to \( \text{IP}(\hat{x}, \mathcal{Y}) \) and obtain \( \text{IP}(\hat{x}, \hat{Y}) \); the individual SECs are similar in form to those of (2.14).

IBDA can use \( \text{MP}(\hat{X}, \hat{A}) \), \( \text{IP}(\hat{X}, \hat{y}) \), and \( \text{IP}(\hat{x}, \hat{Y}) \) to solve \( \text{IP}(\mathcal{X}, \mathcal{Y}) \) by iteratively
(i) solving \( \text{MIP}(\mathcal{X}, \mathcal{Y}) \) with BDA to provide a lower bound on the \( \text{IP}(\mathcal{X}, \mathcal{Y}) \) objective function value, (ii) generating corresponding integer solutions to provide an upper bound, (iii) eliminating the integer solutions found during the current iteration, and (iv) checking for convergence and terminating if the upper and lower bounds are sufficiently close. The method is formalized in Figure 2.2.

This seems like a potentially useful method for solving equivalent deterministic formulations of stochastic integer programs, but we do not find it in the literature. That is, for a
Figure 2.2: Integer Benders decomposition algorithm (IBDA).

Input: An instance of IP(\mathcal{X}, \mathcal{Y}) and allowable optimality gap \varepsilon \geq 0.

Output: An \varepsilon-optimal solution to IP(\mathcal{X}, \mathcal{Y}).

Note: \tilde{z} and \bar{z} are lower and upper bounds on z_{xy}^*; (x^*, y^*) is the incumbent IP(\mathcal{X}, \mathcal{Y}) solution; and \hat{A} is a set of extreme-point vectors for \{ (\alpha, \beta) | \alpha D - \beta \leq f, \alpha \geq 0, \beta \geq 0 \}.

Initialization
1: Relax IP(\mathcal{X}, \mathcal{Y}) to MIP(\mathcal{X}, \mathcal{Y}); decompose MIP(\mathcal{X}, \mathcal{Y}) to MP(\mathcal{X}, \hat{A}) and SUB(\hat{x}, \mathcal{Y});
2: z ← −\infty; \bar{z} ← \infty; \hat{X} ← X; \hat{Y} ← Y; x^*, y^*, \hat{A}_0 ← \emptyset;

Mixed-Integer Relaxation
3: if (MP(\hat{X}, \hat{A}) is infeasible) \{ Go to Step 20; \} /* SECs can make master problem infeasible */
4: Solve MP(\hat{X}, \hat{A}) with BDA for (\hat{x}, \hat{y}) and objective value \hat{z};
5: if (\hat{y} is integer) then
6: if (\hat{z} < \bar{z}) \{ (\bar{z}, x^*, y^*) ← (\hat{z}, \hat{x}, \hat{y}); \} Go to Step 20;
endif
7: if (\hat{z} \geq \bar{z}) \{ Go to Step 20; \} /* Stop if incumbent is better than MP(\hat{X}, \hat{A}) optimum */
8: z ← \hat{z};

Integer Program with Fixed \hat{x}
9: Solve IP(\hat{x}, \hat{Y}) for \hat{y} and objective value \hat{z};
10: \hat{X} ← \hat{X} \setminus \{ \hat{x} \};
11: if (IP(\hat{x}, \hat{Y}) is infeasible) \{ Go to Step 3; \}
12: if (\hat{z} < \bar{z}) \{ (\bar{z}, x^*, y^*) ← (\hat{z}, \hat{x}, \hat{y}); \}
13: if (\hat{z} - \bar{z} \leq \varepsilon z) \{ Go to Step 20; \} /* We assume \bar{z}, z \geq 0 */

Integer Program with Fixed \hat{y}
14: Solve IP(\hat{X}, \hat{y}) for \hat{x} and objective value \hat{z};
15: \hat{Y} ← \hat{Y} \setminus \{ \hat{y} \};
16: if (IP(\hat{X}, \hat{y}) is infeasible) \{ Go to Step 3; \}
17: if (\hat{z} < \bar{z}) \{ (\bar{z}, x^*, y^*) ← (\hat{z}, \hat{x}, \hat{y}); \}
18: if (\hat{z} - \bar{z} \leq \varepsilon z) \{ Go to Step 20; \}
19: Go to Step 3;

Print Solution
20: if (x^*, y^*) \neq \emptyset then Print ("\varepsilon-optimal solution to IP(\mathcal{X}, \mathcal{Y}) is", (\bar{z}, x^*, y^*)); Stop.
stochastic integer program, (i) solve for a first-stage solution $\hat{x}$ with continuous second-stage variables $\hat{y}^1, \ldots, \hat{y}^S$, where superscripts index the $S$ scenarios, (ii) fix $\hat{x} \equiv \hat{x}$ and solve for corresponding $\hat{y}^1, \ldots, \hat{y}^S$, (iii) check bounds, and (iv) if the bounds are not good enough, then add an SEC to eliminate $\hat{x}$ and repeat the process.

BDA assumes that an optimal solution to $\text{MIP}(\mathcal{X}, \mathcal{Y})$ exists and, thus, an optimal solution to $\text{MP}(\mathcal{X}, \hat{\mathcal{A}})$ exists. However, IBDA applies BDA to $\text{MP}(\hat{\mathcal{X}}, \hat{\mathcal{A}})$, which will be infeasible if the set of SECs eliminates all feasible $\hat{x}$, that is, if $\hat{\mathcal{X}} = \emptyset$. Since IBDA confirms the feasibility of $\text{MP}(\hat{\mathcal{X}}, \hat{\mathcal{A}})$ before applying BDA, that algorithm requires no change.

We now demonstrate that IBDA terminates in a finite number of iterations, and correctly solves $\text{IP}(\mathcal{X}, \mathcal{Y})$. We formalize finiteness and correctness with Theorems 2.1 and 2.2, respectively.

**Theorem 2.1** IBDA is finite.

**Proof.** During all but the final iteration, Step 10 of IBDA is guaranteed to introduce an SEC that eliminates a partial solution, i.e., $\hat{x}$. Since $|\mathcal{X}| = 2^{n_x}$, IBDA requires, at most, $2^{n_x}$ iterations before $\text{MP}(\hat{\mathcal{X}}, \hat{\mathcal{A}})$ is infeasible because all partial solutions of form $\hat{x}$ have been eliminated with SECs.

**Theorem 2.2** IBDA correctly solves $\text{IP}(\mathcal{X}, \mathcal{Y})$.

**Proof.** Let $(x^*, y^*)$ be the optimal solution to a feasible instance of $\text{IP}(\mathcal{X}, \mathcal{Y})$; let $(\hat{x}^*, \hat{y}^*)$ be the incumbent integer solution if one has been found; let $(\hat{x}, \hat{y})$ be the solution to $\text{MIP}(\mathcal{X}, \mathcal{Y})$ returned by BDA; and let $z(x, y)$ denote the objective function value for any solution $(x, y)$ to either $\text{IP}(\mathcal{X}, \mathcal{Y})$ or $\text{MIP}(\mathcal{X}, \mathcal{Y})$. If $\text{IP}(\mathcal{X}, \mathcal{Y})$ is feasible and IBDA terminates before finding $(x^*, y^*)$, then one of four statements must be true:

1. In Step 3, during the first IBDA iteration, $\text{MP}(\hat{\mathcal{X}}, \hat{\mathcal{A}})$ is infeasible. However, since $\text{IP}(\mathcal{X}, \mathcal{Y})$ is feasible, $\text{MP}(\hat{\mathcal{X}}, \hat{\mathcal{A}})$ necessarily has a feasible solution during the first IBDA iteration, that is, $\hat{x} = x^*$. 

17
2. In Steps 5 and 6, $\hat{y}$ is integer and $z(x^*, y^*) < z(\hat{x}, \hat{y}) < z(\check{x}, \check{y})$. However, BDA cannot return an integer $(\hat{x}, \hat{y})$ unless either $z(\hat{x}, \hat{y}) = z(x^*, y^*)$ or the incumbent integer solution is $(x^*, y^*)$.

3. In Step 7, $z(\hat{x}, \hat{y}) \geq z(\check{x}, \check{y}) > z(x^*, y^*)$. However, if $z(x^*, y^*) < z(\check{x}, \check{y})$, then (i) $x^* \in \hat{X}$ and $y^* \in \hat{Y}$, (ii) $(x^*, y^*)$ is a feasible solution to $\text{MP}(\hat{X}, \hat{A})$, and (iii) $z(\hat{x}, \hat{y}) \leq z(x^*, y^*)$.

4. In either Step 13 or Step 18, $\bar{z} - z \leq \varepsilon \bar{z}$ and $z(\hat{x}, \hat{y}) - z(x^*, y^*) > \varepsilon z(x^*, y^*)$. However, $\bar{z} - z > \varepsilon \bar{z}$ since $\bar{z} = z(\hat{x}, \hat{y})$ and $z(x^*, y^*) \geq z$.

During an iteration, IBDA will (i) apply BDA to obtain $(\hat{x}, \hat{y})$, (ii) solve $\text{IP}(\hat{x}, \hat{y})$ for $(\hat{x}, y^*(\hat{x})) = (\hat{x}, \hat{y})$, and (iii) solve $\text{IP}(\hat{x}, \hat{y})$ for $(x^*(\hat{y}), \hat{y})$. Solving $\text{IP}(\hat{x}, \hat{y})$ is not essential, however. Instead, IBDA could simply solve $\text{MIP}(\hat{X}, \hat{Y})$ for $\hat{x}$, determine the corresponding $\hat{y}$, augment $\text{MIP}(\hat{X}, \hat{Y})$ with an SEC for $\hat{x}$, and then re-solve $\text{MIP}(\hat{X}, \hat{Y})$. In fact, for certain problems, solving $\text{IP}(\hat{x}, \hat{y})$ could increase overall solution times. We expect, however, that for at least some problems, solving $\text{IP}(\hat{x}, \hat{y})$ would reduce the number of IBDA iterations and improve overall solution times.

2.5 Set-Packing Problem

We test IBDA on the set-packing problem ($\text{PACK}$). $\text{PACK}$ is widely studied (Torres, 2003), and has numerous practical applications such as cargo-ship scheduling (Fisher and Rosenwein, 1989), communication-spectrum licensing (Günlük et al., 2005), and railway-infrastructure planning (Gandibleux et al., 2005). While the IBDA development assumes a minimizing IP, IBDA and BDA can be readily adapted to solve maximizing IPs and MIPs, respectively. Thus, we introduce ($\text{PACK}$) in its common form (e.g., Balas and Padberg, 1976; Delorme et al., 2004).

Excluding constraints (2.18), $\text{PACK}^+(\mathcal{X}, \mathcal{Y})$ represents a standard set-packing problem whose variables have been arbitrarily partitioned into two sets:
\[
\text{PACK}^+(\mathcal{X}, \mathcal{Y}) \quad z^*_x = \max_{x \in \mathcal{X}, y \in \mathcal{Y}} cx + fy \\
\text{s.t. } Bx \leq 1 \\
Bx + Dy \leq 1,
\]

where \( \mathcal{X} \equiv \{ x \in \{0, 1\}^{nx} \}; \mathcal{Y} \equiv \{ y \in \{0, 1\}^{ny} \}; B \) and \( D \) are 0-1 matrices with dimension \( m \times n_x \) and \( m \times n_y \), respectively; and the vectors \( c, f, x, y, \) and \( 1 \) conform. Together, constraints (2.19) and \( x \in \mathcal{X} \) jointly ensure that when \( y \) is relaxed, \( y \leq 1 \). Given this implicit upper bound on \( y \), (i) the Benders decomposition subproblem does not require constraints (2.9), (ii) the extreme-point vectors in (2.12) take the form \( \hat{\alpha} \), rather than \( (\hat{\alpha}, \hat{\beta}) \), and (iii) the subproblem polytope, for a particular \( \hat{x} \) is \( \mathcal{Y}(\hat{x}) = \{ y \in \mathcal{R}_+^{ny} | Dy \leq 1 - B\hat{x} \}. \)

Constraints (2.18) are clearly redundant, but they ensure that Assumption 2.1 holds and that IBDA solves \( \text{PACK}^+(\mathcal{X}, \mathcal{Y}) \). In Section 2.6, we will use \( \text{PACK}(\mathcal{X}, \mathcal{Y}) \) to denote \( \text{PACK}^+(\mathcal{X}, \mathcal{Y}) \) without these redundant constraints.

### 2.5.1 Instances that Yield Totally Unimodular Subproblems

In general, we prefer IPs of form \( \text{IP}(\mathcal{X}, \mathcal{Y}) \), for which \( \mathcal{Y}(\hat{x}) \) is an integer polytope for any fixed, feasible \( \hat{x} \). Therefore, we first examine set-packing problems with TU \( D \) matrices. Given any fixed \( \hat{x} \in \{0, 1\}^{nx} \) and a 0-1 \( B \)-matrix, \( 1 - B\hat{x} \) is a 0-1 vector and \( \mathcal{Y}(\hat{x}) \) has integral extreme points provided every square sub-matrix of \( D \) has determinant equal to 0, \pm 1 (Hoffman and Kruskal, 2010), that is, \( D \) is TU (Conforti et al., 2006). A subproblem instance with such a polyhedron must yield an integer optimal solution (De Werra, 1981). Several methods exist to determine whether a matrix such as \( D \) is TU. For example, a matrix with two 1’s per column is TU if its rows can be partitioned into two sets such that the 1’s in each column are in different sets (Tamir, 1976; Hoffman and Kruskal, 2010).

In Section 2.6, we test IBDA on set packing problems that yield TU subproblem constraint matrices. Ideally, given any instance of PACK, we would like to create a partition of the instance’s variables to yield \( \text{PACK}^+(\mathcal{X}, \mathcal{Y}) \) such that Proposition 2.2 holds. However, as a
proof of concept, we will simply generate instances of $\text{PACK}^+(\mathcal{X}, \mathcal{Y})$ with $D$ matrices that either represent bipartite networks or exhibit the consecutive 1’s property (COP). These will be defined in Section 2.6.

Consider a special case of $\text{PACK}^+(\mathcal{X}, \mathcal{Y})$, in which $D$ has two non-zero coefficients per column and is the node-edge incidence matrix of a bipartite graph. Since this matrix is totally unimodular (Goldberg et al., 1992), Proposition 2.2 holds and BDA solves $\text{PACK}^+(\mathcal{X}, \mathcal{Y})$.

Proposition 2.2 also holds when the 0-1 $D$-matrix has the COP (Ruf and Schöbel, 2004), that is, there exists a permutation of the rows that places the 1’s consecutively in each column (Booth and Lueker, 1976). For obvious computational reasons, we prefer a partition which maximizes the number of columns in $D$, although finding such a partition is an NP-hard problem (e.g., Hajiaghayi and Ganjali, 2002; Pierce and Winfree, 2002).

2.5.2 Nonbipartite Maximum Matching

In what follows, “maximum matching” refers only to nonbipartite maximum matching. Let $G = (V, E)$ be an undirected graph with vertex set $V$ and edge set $E$, where each edge is unweighted and connects distinct vertices $u, v \in V$. While the polynomial “blossom” algorithm solves this maximum cardinality matching problem, that algorithm has poor worst-case complexity (Edmonds, 1965; Kolmogorov, 2009). Alternatively, Balas and Padberg (1976) formulate the maximum-matching problem as an IP, which, for a given partitioning of the edges, can be written in the form of $\text{PACK}(\mathcal{X}, \mathcal{Y})$. The maximum cardinality matching problem extends easily to the maximum weighted matching, where each edge has a weight $c_E$. (Note that we do not generate any new constraints to help define the IP polytope as Balas and Padberg 1976 do.)

IBDA correctly solves the maximum-matching problem for an arbitrary edge partitioning. But, if $D$ represents a bipartite network, then Proposition 2.2 holds and BDA suffices. We are interested in maximum weighted matchings in complete graphs because such matchings have applications in anomaly detection (Ruth and Koyak, 2011). Consequently, we test IBDA on this class of problems.
2.6 Computational Results

This section presents computational experiments that compare solving $\text{PACK}^+(X, Y)$ with IBDA to solving $\text{PACK}(X, Y)$ by direct application of the CPLEX branch-and-bound algorithm, a method that we will refer to as “B&B.” (Recall that $\text{PACK}(X, Y)$ is the problem $\text{PACK}^+(X, Y)$ without constraints (2.18).) For all testing, we generate problems in AMPL Version 20120804 and solve with CPLEX solver 12.4 on a computer with 16 Intel Xeon E5520 quad-core processors and 12GB of memory. We first test IBDA and B&B on deliberately constructed set-packing instances. For all tests, we use default CPLEX settings, except as otherwise noted.

We conclude by briefly introducing the facility-location problem from the Appendix, and presenting computational results for small instances.

2.6.1 Instances with a Bipartite-Graph Subproblem

Given a sequence of numerically encoded observations, $\theta_1, \ldots, \theta_m$, generated by some process, Ruth and Koyak (2011) construct a complete graph with one vertex per observation, $\theta_i$, one edge variable per pair of observations, $\theta_i$ and $\theta_i'$, and edge weights that represent the “distance” between observations $\theta_i$ and $\theta_i'$. The authors then solve a weighted non-bipartite matching as part of a nonparametric test for heterogeneity of the process.

With this application in mind, we first compare IBDA to B&B by solving for maximum-weight matchings on a 1000-node complete graph, $G = (V, E)$. For these problems, the $i^{th}$ row of $B$ and the $i^{th}$ row of $D$ jointly represent vertex $i$, while each matrix column represents a distinct graph edge connecting vertices $i$ and $i'$. We generate $D$ to represent the largest possible bipartite graph in $G = (V, E)$, that is, a graph in which every cycle has an even number of edges (Edmonds, 1965). The matrix $B$ represents a graph containing the remaining edges. Figure 2.3 illustrates such a partition. For a 1000-node graph, $B$ and $D$ have 249,500 and 250,000 columns, respectively.
Using both IBDA and B&B, we investigate 15 groups of problems, each defined by the width of the uniform distribution from which the “observations” are drawn and by the optimality tolerance used for both IBDA and B&B. We draw each vertex observation $\theta_i$ from the uniform distribution $U(0.5 - \delta, 0.5 + \delta)$, for $\delta \in \{0.5, 0.1, 0.01, 0.001, 0.0001\}$, and define $c_{i,i'} = |\theta_i - \theta_{i'}|$. We test relative optimality tolerances of 0.001, 0.01, 0.05. Based on initial test results, we specify the primal simplex method for the resulting Benders subproblems.

Since $D$ is the node-edge incidence matrix of a bipartite graph, Proposition 2.2 holds and BDA yields an integer solution, which provides IBDA a potential advantage over B&B. After solving $\text{MIP}(\mathcal{X}, \mathcal{Y})$ with BDA, IBDA immediately returns an optimal solution to $\text{IP}(\mathcal{X}, \mathcal{Y})$, with no need to explicitly solve for an integer $y$. B&B may also benefit from the TU $D$-matrix, provided an appropriate branching priority is employed. If CPLEX branches on a fractional $x$-variable at every non-integer node in the branch-and-bound tree, then B&B will obtain an integer solution to $\text{IP}(\mathcal{X}, \mathcal{Y})$ with no need to branch on any $y$-variable. Therefore, we prioritize branching on the $x$-variables when solving $\text{PACK}(\mathcal{X}, \mathcal{Y})$ with B&B.

We implement IBDA using AMPL, which incurs some computational overhead. Since much of this overhead could be avoided by implementing IBDA in a compiled language such as C++, the solution times reported here only include the CPU time required by CPLEX.

![Diagram](image.png)

**Figure 2.3:** Partitioning a complete graph into a largest possible bipartite graph, represented by $D$, and a non-bipartite graph, represented by $B$. 
We note that including the AMPL overhead would not change the relative ranking of the two methods, however. For each group of problems, Table 2.1 reports the average solution times for ten random instances using each solution method.

With a relative optimality tolerance of 0.001, IBDA reduces solution times by 23-78% compared to B&B; with a relative optimality tolerance of either 0.01 or 0.05, the reductions are greater than 90%. We find that IBDA solution speeds improve significantly for cases in which the optimality tolerance is 0.01 or 0.05; for these cases, a single BDA iteration suffices to prove $\varepsilon$-optimality. B&B solution times do not improve significantly with relaxed optimality tolerance, however, which appears to be the result of difficulty (i) solving the IP’s LP relaxation at the root node of the branch-and-bound tree, and (ii) generating an initial integer solution.

### 2.6.2 Instances with Consecutive 1’s Property in the Subproblem Constraint Matrix

Our second IBDA test uses a set-packing problem whose matrix $D$ has the COP. We generate a $100 \times 1000$ $B$-matrix with $P(B_{ij} = 1) = p \ orall \ i, j$, where $p$ represents the expected

<table>
<thead>
<tr>
<th>Distribution of observations</th>
<th>$\varepsilon = 0.001$</th>
<th>Relative Optimality Tolerance</th>
<th>$\varepsilon = 0.01$</th>
<th>$\varepsilon = 0.05$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IBDA</td>
<td>B&amp;B</td>
<td>IBDA</td>
<td>B&amp;B</td>
</tr>
<tr>
<td>Unif(0, 1)</td>
<td>17.2 22.0 1.7</td>
<td>48.5 28.7</td>
<td>2.6 0.7 1.0</td>
<td>39.4 4.7</td>
</tr>
<tr>
<td>Unif(0.4, 0.6)</td>
<td>14.3 24.7 1.5</td>
<td>64.4 9.9</td>
<td>1.9 0.8 1.0</td>
<td>69.5 12.7</td>
</tr>
<tr>
<td>Unif(0.49, 0.51)</td>
<td>30.0 26.4 2.2</td>
<td>38.8 2.6</td>
<td>3.2 0.5 1.0</td>
<td>38.4 2.9</td>
</tr>
<tr>
<td>Unif(0.499, 0.501)</td>
<td>8.8 14.1 1.3</td>
<td>36.9 2.5</td>
<td>3.2 0.5 1.0</td>
<td>38.0 3.8</td>
</tr>
<tr>
<td>Unif(0.4999, 0.5001)</td>
<td>26.7 45.3 1.9</td>
<td>39.2 9.7</td>
<td>2.4 0.8 1.0</td>
<td>87.9 77.7</td>
</tr>
</tbody>
</table>

**Notes:** CPLEX solves the monolithic MIP with a relative optimality tolerance of $\varepsilon$, and solves IBDA master problems using its default relative optimality tolerance of $10^{-4}$.

Bold font indicates the fastest average solution time for each combination of distribution and $\varepsilon$. 

23
density of the resulting matrix \( B \). The \( D \)-matrix is based upon an integer parameter \( \{ \tau \mid \tau \geq 2 \} \). For \( r \in \{2, \ldots, \tau \} \), \( D \) contains all distinct columns that have \( r \) ones and exhibit the COP. Figure 2.4 illustrates a \( D \)-matrix with \( \tau = 3 \).

Let \( c_j = 1 \) and \( f_k = 0.01q_k(1 + \tilde{\mu}) \forall k \), where \( q_k \) is the number of ones in column \( k \) and \( \tilde{\mu} \) is a uniform random variable on \([0, 1]\). We investigate groups of problems, each with a different combination of relative optimality tolerance (\( \varepsilon \)), \( p \), and \( \tau \). For B&B, we prioritize branching on the \( x \)-variables as in Section 2.6.1. Table 2.2 provides the average CPU time required by CPLEX for five random instances from each group of problems.

IBDA performs relatively well, compared to B&B, when the relative optimality tolerance is 0.05 and/or the expected density of the \( B \)-matrix is 0.01. For the other groups of problems, individual IBDA master-problem solution times often exceed the average time required for CPLEX to solve the corresponding monolithic MIPs.

Both Table 2.1 and Table 2.2 clearly indicate that, for certain set-packing problems that can be partitioned to yield a TU \( D \)-matrix, IBDA requires less CPU time than B&B. We would also, however, like IBDA to solve general set-packing problems that cannot be partitioned in such a manner. But, since IBDA does not consistently outperform B&B when

\[
D = \begin{bmatrix}
1 & 0 & \ldots & 0 & 0 & 1 & 0 & \ldots & 0 & 0 \\
1 & 1 & \ldots & 0 & 0 & 1 & 1 & \ldots & 0 & 0 \\
0 & 1 & \ldots & 0 & 0 & 1 & 1 & \ldots & 0 & 0 \\
0 & 0 & \ldots & 0 & 0 & 0 & 1 & \ldots & 0 & 0 \\
0 & 0 & \ldots & 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\
0 & 0 & \ldots & 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\
\vdots & \vdots & & & & & & & & \\
0 & 0 & \ldots & 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\
0 & 0 & \ldots & 0 & 0 & 0 & 0 & \ldots & 1 & 0 \\
0 & 0 & \ldots & 1 & 0 & 0 & 0 & \ldots & 1 & 1 \\
0 & 0 & \ldots & 1 & 1 & 0 & 0 & \ldots & 1 & 1 \\
0 & 0 & \ldots & 0 & 1 & 0 & 0 & \ldots & 0 & 1
\end{bmatrix}
\]

Figure 2.4: \( D \)-matrix with consecutive 1’s property, where the maximum number of consecutive 1’s \( \tau = 3 \). The left side of \( D \) contains all possible columns with two consecutive 1’s, while the right side contains all possible columns with three consecutive 1’s.
Table 2.2: Comparison of IBDA and B&B run times for set-packing problems whose $D$-matrix has the COP. For each combination of optimality tolerance ($\varepsilon$), $p$, and $\tau$, we solve five randomly generated $\text{PACK}(\mathcal{X}, \mathcal{Y})$ instances and report the average CPLEX time in seconds.

<table>
<thead>
<tr>
<th>Relative Optimality Tolerance ($\varepsilon$)</th>
<th>$\tau$</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$p$</td>
<td>IBDA</td>
<td>B&amp;B</td>
<td>IBDA</td>
<td>B&amp;B</td>
</tr>
<tr>
<td>0.001</td>
<td>0.01</td>
<td>0.05</td>
<td><strong>0.04</strong></td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>0.001</td>
<td>0.05</td>
<td>7.96</td>
<td><strong>2.55</strong></td>
<td>9.27</td>
<td><strong>3.00</strong></td>
</tr>
<tr>
<td>0.01</td>
<td>0.01</td>
<td>0.06</td>
<td><strong>0.04</strong></td>
<td>0.06</td>
<td><strong>0.05</strong></td>
</tr>
<tr>
<td>0.01</td>
<td>0.05</td>
<td>2.91</td>
<td><strong>1.58</strong></td>
<td>1.23</td>
<td><strong>0.36</strong></td>
</tr>
<tr>
<td>0.05</td>
<td>0.01</td>
<td>0.06</td>
<td><strong>0.04</strong></td>
<td>0.06</td>
<td><strong>0.05</strong></td>
</tr>
<tr>
<td>0.05</td>
<td>0.05</td>
<td>0.11</td>
<td><strong>0.10</strong></td>
<td>0.13</td>
<td><strong>0.10</strong></td>
</tr>
</tbody>
</table>

**Notes:** CPLEX solves the monolithic MIP with a relative optimality tolerance of $\varepsilon$, and solves IBDA master problems using its default relative optimality tolerance of $10^{-4}$. Bold font indicates the fastest average solution time for each group.

Solving IPs that satisfy Proposition 2.2, IBDA is unlikely to outperform B&B when solving IPs that do not. (For these general IPs, IBDA uses BDA to solve a mixed-integer relaxation of the IP; IBDA must then solve for corresponding integer solutions, generate SECs, and potentially perform additional BDA iterations.)

For IBDA to be useful, we must focus on improving BDA’s computational effectiveness. We expect that implementing BDA using a compiled language such as C++ (as described in Section 2.6.3) will reduce overhead and, therefore, overall solution times, compared to our current implementation that relies on a modeling language. More importantly perhaps, we will focus on overcoming relatively slow solution speeds for the Benders master problems. (Recall that for groups of problems shown in Table 2.2, individual master-problem solution times can exceed the average solution times for monolithic MIPs.) Our options include: (i) employing heuristics to identify promising solutions as a means to reduce iterations; (ii) implementing integer cutting planes to generate integer subproblem solutions and to tighten the master problem’s LP relaxation; and (iii) developing alternate methods to solve the Benders master problems.

In Chapters 3 and 4, we develop a BDA that solves master problems using explicit enumeration and demonstrate its effectiveness by solving a stochastic capacitated facility-
location problem. The next section provides brief computational results using this new implementation.

2.6.3 Practical Application with an Alternate Master-Problem Solution Technique

We conclude by examining an integer variant of the stochastic capacitated facility-location problem from Sanchez and Wood (2006), in which customer demands for a single product are uncertain. Unlike Sanchez and Wood, we restrict all demands to integer values as in Melkote and Daskin (2001). We present a brief description here; the appendix provides a complete formulation.

The stochastic capacitated facility-location problem (SCFL) describes a two-part decision problem for a manufacturing company: whether or not to construct facilities at each location in a predetermined set of candidate sites $I$; and, after realizing customer demands, how to serve those demands at minimum cost from the constructed facilities, while paying a penalty for each unit of unserved demand. The manufacturer’s objective is to minimize the facility-construction costs plus the expected costs of meeting customer demands, to include any penalties. Sanchez and Wood describe their test problems using four key characteristics: (i) $I$ is the number of candidate sites, (ii) $J$ is the number of customers, (iii) $b$ is the maximum number of facilities that may be constructed, (iv) and $\beta$ controls the variance of the demand distribution. For a given test problem $P(I, J, b, \beta, |S|)$, a model instance also includes location, capacity, and cost parameters along with $|S|$ scenario demands for each customer $d_j$, where $S$ denotes a set of scenarios. For the following tests, these demands are drawn randomly from the set $\{1, 2, 3\}$.

When we apply Benders decomposition to SCFL, the resulting master problem selects a facility-construction plan $\hat{x}$, while the individual subproblems (one per scenario) optimize distribution based on $\hat{x}$. Sanchez and Wood use uncapacitated arcs. Without loss of generality, let each arc have a maximum capacity of 3, that is, the maximum demand for any customer. The resulting subproblem is a network flow problem with integer arc capacities.
and integer supplies and demands. Thus, Proposition 2.2 holds and BDA will solve SCFL.

For these tests, we implement IBDA in C++ using eight Intel Xeon E5520 quad-core processors on a computer with 12 GB of memory. We let $\varepsilon = 0.01$ and solve master problems and subproblems using CPLEX solver 12.4 (IBM, 2011) with default parameter settings except that we specify eight parallel threads and deterministic parallelism. For comparison, we solve the monolithic MIP with CPLEX, using a relative optimality tolerance of 0.01.

We investigate the fifteen groups of problems from Sanchez and Wood (2006) by solving ten random, 50-scenario instances from each. Table 2.3 shows average solution times and related statistics to the left of the double vertical lines for both B&B and IBDA. Note that these values represent total elapsed solution time, rather than CPLEX times as reported in the preceding tables.

IBDA is faster than B&B for the groups with fewer sites and fewer customers, provided the number of Benders iterations remains “relatively low.” For instances that require a large number of iterations, master-problem size may be limiting IBDA speed. Magnanti and Wong (1981) note that the master problem can be a “major computational bottleneck” for BDA.

Table 2.3: Statistics for IBDA and B&B solving ten random instances of the stochastic facility location problem $S(I, J, b, \beta, 50)$ using relative optimality tolerance $\varepsilon = 0.01$.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
<td>5</td>
<td>0.1</td>
<td>0.47</td>
<td>0.29</td>
<td>0.33</td>
<td>0.28</td>
<td>12.1</td>
<td></td>
<td>0.18</td>
<td>0.13</td>
<td>12.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>0.56</td>
<td>0.40</td>
<td>0.31</td>
<td>0.16</td>
<td>12.8</td>
<td></td>
<td>0.18</td>
<td>0.08</td>
<td>12.8</td>
<td></td>
<td>0.13</td>
<td>0.07</td>
<td>12.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.4</td>
<td>0.67</td>
<td>0.64</td>
<td>0.20</td>
<td>0.12</td>
<td>10.0</td>
<td></td>
<td>0.13</td>
<td>0.07</td>
<td>10.0</td>
<td></td>
<td>1.51</td>
<td>1.15</td>
<td>9.2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>40</td>
<td>8</td>
<td>0.1</td>
<td>2.39</td>
<td>1.33</td>
<td>5.04</td>
<td>3.67</td>
<td>49.2</td>
<td></td>
<td>3.76</td>
<td>0.95</td>
<td>3.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.2</td>
<td>2.19</td>
<td>1.36</td>
<td>0.1</td>
<td>1.21</td>
<td>1.56</td>
<td></td>
<td>0.55</td>
<td>0.40</td>
<td>16.0</td>
<td></td>
<td>1.86</td>
<td>1.48</td>
<td>65.1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.4</td>
<td>1.88</td>
<td>1.09</td>
<td>1.09</td>
<td>8.15</td>
<td>8.63</td>
<td>65.4</td>
<td></td>
<td>3.38</td>
<td>2.87</td>
<td>65.4</td>
<td></td>
<td>1.86</td>
<td>1.48</td>
<td>65.1</td>
</tr>
<tr>
<td>7</td>
<td>0.1</td>
<td>0.79</td>
<td>0.23</td>
<td>0.17</td>
<td>0.06</td>
<td>4.8</td>
<td></td>
<td>0.14</td>
<td>0.06</td>
<td>4.8</td>
<td></td>
<td>0.14</td>
<td>0.06</td>
<td>4.8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.2</td>
<td>1.10</td>
<td>0.44</td>
<td>0.83</td>
<td>1.16</td>
<td>12.6</td>
<td></td>
<td>0.47</td>
<td>0.42</td>
<td>12.6</td>
<td></td>
<td>0.33</td>
<td>0.31</td>
<td>10.2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.4</td>
<td>0.98</td>
<td>0.33</td>
<td>0.54</td>
<td>0.62</td>
<td>10.2</td>
<td></td>
<td>0.33</td>
<td>0.31</td>
<td>10.2</td>
<td></td>
<td>0.33</td>
<td>0.31</td>
<td>10.2</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>18</td>
<td>30</td>
<td>10</td>
<td>0.1</td>
<td>0.78</td>
<td>0.35</td>
<td>0.35</td>
<td>1.79</td>
<td>13.8</td>
<td></td>
<td>0.12</td>
<td>0.12</td>
<td>13.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.2</td>
<td>0.84</td>
<td>0.32</td>
<td>1.25</td>
<td>1.57</td>
<td>17.5</td>
<td></td>
<td>0.60</td>
<td>0.39</td>
<td>17.5</td>
<td></td>
<td>0.55</td>
<td>0.60</td>
<td>18.1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.4</td>
<td>0.84</td>
<td>0.50</td>
<td>1.53</td>
<td>2.76</td>
<td>18.7</td>
<td></td>
<td>0.55</td>
<td>0.60</td>
<td>18.1</td>
<td></td>
<td>0.55</td>
<td>0.60</td>
<td>18.1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>20</td>
<td>50</td>
<td>10</td>
<td>0.1</td>
<td>4.01</td>
<td>3.35</td>
<td>0.69</td>
<td>8.70</td>
<td>40.8</td>
<td></td>
<td>0.23</td>
<td>0.23</td>
<td>40.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.2</td>
<td>4.04</td>
<td>2.30</td>
<td>7.82</td>
<td>6.66</td>
<td>56.3</td>
<td></td>
<td>3.00</td>
<td>1.77</td>
<td>58.4</td>
<td></td>
<td>4.46</td>
<td>3.30</td>
<td>89.8</td>
<td></td>
</tr>
</tbody>
</table>

Notes: $\varepsilon = 0.01.$ CPLEX solves the monolithic MIP with a relative optimality tolerance of $\varepsilon$, and solves IBDA master problems using its default relative optimality tolerance of $10^{-4}$. For each group, bold font indicates the fastest average solution time from among B&B and IBDA.
During each iteration, the new Benders cut improves the approximation of \( \text{MP}(X, A) \), but the resultant master problem is larger and potentially more difficult to solve (Naoum-Sawaya and Elhedhli, 2013). As Holmberg (1994) notes, increasingly difficult master problems can even render BDA too slow for practical use.

In Chapters 3 and 4, we develop and apply an implementation of Benders decomposition that solves master problems by enumerating all feasible solutions. This implementation stores objective-function values for all master-problem solutions, which ensures that the computational effort needed to solve the master problem does not increase from one iteration to the next. We solve each of the SCFL model instances described above using the enumerative IBDA and report solution statistics to the right of the double vertical lines in Table 2.3. Enumerative IBDA proves to be faster than the standard implementation. Additionally, enumerative IBDA solution times are smaller than B&B solution times for each group of problems. Since this new implementation is clearly faster then B&B, IBDA has the potential to outperform B&B for certain IPs that do not satisfy Proposition 2.1. One candidate may be the stochastic facility-location problem with sole-sourcing described by Silva and Wood (2006).

2.7 Conclusions

We have developed a new integer Benders decomposition algorithm (IBDA) that extends Benders decomposition to pure integer programs (IPs) with 0-1 vectors \( x \) and \( y \). IBDA temporarily relaxes \( y \) to continuous \( y \) and solves the resulting mixed-integer program with Benders decomposition algorithm (BDA) for \((x^*, y^*)\). If BDA does not yield an integer \( y^* \), then we (i) fix \( \hat{x} \equiv x^* \), (ii) solve \( \text{IP}(\hat{x}, \hat{y}) \) for an integer optimal \( \hat{y} \), (iii) solve \( \text{IP}(\hat{x}, \hat{y}) \) for \( \hat{x} \), (iv) eliminate the most recent integer solutions by adding solution-elimination constraints to \( \text{MP}(\hat{x}, \hat{A}) \), \( \text{IP}(\hat{x}, \hat{y}) \), and \( \text{IP}(\hat{x}, \hat{y}) \), and (v) resume Benders decomposition iterations.

Testing on several set-packing instances indicates that IBDA can be competitive with direct solution via a standard branch-and-bound (B&B) algorithm. In particular, IBDA solves certain problems of the form \( \text{PACK}^+(X, Y) \), each of which has a totally-unimodular
$D$-matrix, faster than a standard B&B algorithm solves the corresponding $\text{PACK}(\mathcal{X}, \mathcal{Y})$. We also briefly show the potential of an IBDA method that solves master problems by enumerating all feasible master-problem solutions. (Chapters 3 and 4 develop the enumerative method.)

Future work will improve the implementation of IBDA by employing the enumerative solution technique of Chapters 3 and 4. We will test this new implementation on promising IPs, such as the stochastic facility-location problem with sole-sourcing.
CHAPTER 3
USING ENUMERATION TO SOLVE BINARY MASTER PROBLEMS IN BENDERS DECOMPOSITION

Modified from a paper that is in review as of February 3, 2014
D. Antony Tarvin, R. Kevin Wood, Alexandra M. Newman

3.1 Abstract

We show how to solve certain mixed-integer programs using a version of Benders decomposition that solves each master problem by explicit enumeration. By storing objective-function values for all master-problem solutions, calculations from one iteration to the next are simple. Empirical speedup in solving the master problem is almost linear with the number of parallel processors. Computational tests on a sequence of master problems show order-of-magnitude reductions in solution times compared to solving those master problems with parallel branch-and-bound.

3.2 Introduction

Many mixed-integer programs (MIPs) solve very efficiently through decomposition (e.g., Ford, Jr. and Fulkerson, 1958; Van Slyke and Wets, 1969; Sen and Sherali, 2006). Benders decomposition is typical (Benders, 1962) and is employed in a variety of applications such as distribution planning (Geoffrion and Graves, 1974), power-flow optimization (Alguacil and Conejo, 2000), and preventive-maintenance scheduling (Canto, 2008).

Magnanti and Wong (1981) note that a key computational challenge for the Benders decomposition algorithm (BDA) is the need to solve the mixed-integer master problem repeatedly. Each (major) iteration of BDA introduces a new constraint, a “Benders cut,” that increases the master problem’s size and, thus, its worst-case solution time (Nemhauser and
Wolsey, 1999, p. 125). A typical BDA implementation solves the master problems by standard, linear-programming-based branch-and-bound (B&B). (Note that the current chapter does not consider “branch-and-cut” methods, which add Benders cuts while traversing the B&B tree of a single master problem; for example, see Naoum-Sawaya and Elhedhli 2013.)

Several key techniques that modify the nominal master problem can improve BDA’s computational efficiency. Master-problem cuts can be tightened by exploiting (i) multiple optimal dual solutions from subproblems and/or (ii) problem-specific information (Magnanti and Wong, 1981; Wentges, 1996). Another technique, “cut dropping,” yields smaller master problems by deleting “unattractive” Benders cuts: typically, these are slack cuts from early iterations, which may be unnecessary to ensure convergence (e.g., Van Dinter et al., 2013). The resulting, smaller master problems solve more quickly and, consequently, BDA’s performance may improve if (i) most of the unattractive cuts are, in fact, unnecessary, and (ii) any necessary cuts that are dropped can be regenerated efficiently. Finally, we note that if fixing integer variables yields independent subproblems, then the “multicut” master problem of Birge and Louveaux (1988) applies. This reformulation is well-known to converge more quickly than the standard, “single-cut” formulation for the same problem. While our test problem does not have independent subproblems, we discuss how our approach could solve a multicut master problem.

Our interests do not lie in modifying master-problem formulations, however. Rather, we seek a new, faster solution method that applies to standard master-problem formulations, formulations with tightened cuts, and perhaps even to multicut formulations. Several specialized solution techniques appear in the literature already.

Geoffrion and Graves (1974) suboptimize the Benders master problem. That is, they stop the master-problem solution process after obtaining a feasible integer solution whose objective value incrementally improves upon the incumbent value. This technique is convergent and does not waste time optimizing initial master problems which, typically, are less informative than those in later iterations. It may increase the number of iterations, however, and
A user could also treat the unattractive Benders cuts mentioned above as “lazy constraints,” and let an enhanced B&B solution algorithm add these cuts on an as-needed basis. The lazy-constraint technique has proven useful in solving MIPs directly (Tóth et al., 2013), and it might help with solving a Benders master problem. Specifically, an unattractive cut would be labeled a “lazy cut,” omitted from the master problem’s constraint matrix, monitored in a computationally efficient manner, and reintroduced into the constraint matrix if needed for convergence (for example, see IBM 2011). So, unlike cut dropping, this technique would not change the master problem’s formulation. Although Wu (2013) mentions the possibility of using lazy constraints in a Benders-based method, this author provides no details and we find no applications to Benders decomposition in the literature. In Section 3.5.2, we compare lazy constraints to explicit enumeration as a device for enhancing efficiency.

Tuy and Horst (1988) describe a “restart B&B algorithm” that supports decomposition approaches to solving certain global-optimization problems. These authors modify the conventional B&B fathoming criteria to ensure that a master-problem’s B&B tree can be stored and used to provide a “warm start” in the subsequent iteration of a decomposition algorithm. We must presume, however, that the technique has limited value within a standard BDA, as the method has been available for over 20 years and yet we find no applications to BDA in the literature.

Salmerón and Wood (2014) first suggest the solution technique we pursue in the current chapter: solve the master problem in BDA by explicit enumeration. This technique has clear computational limitations, but we find important applications, especially in the area of infrastructure protection and interdiction, as explained in Section 3.5; see also Brown et al. (2006) and the references therein. The implementation in Salmerón and Wood (2014) is not consistently faster than standard BDA, but those authors restrict themselves to (i) programming the enumeration using a slow algebraic modeling language, and (ii) implementing only a serial algorithm. We use C++ to implement a fast, parallel enumeration algorithm.
Many authors note that an efficient integer-programming algorithm must employ effective implicit enumeration. That is, it must exclude a substantial number of dominated solutions (e.g., Geoffrion, 1967). Despite this, explicit enumeration of master-problem solutions within BDA may be reasonable. Benders decomposition views the MIP as a problem of optimizing a function of the integer variables while indirectly determining the values of the continuous variables. We may not be able to optimize this function by evaluating it for all feasible settings of integer variables, because each such function evaluation requires the solution of a substantial linear program. But, every iteration of BDA yields a successively improving approximation of the function, which can be evaluated quickly and, therefore, may be well suited to optimization by enumeration, that is, by evaluating the approximation for every feasible setting of integer variables and simply selecting as optimal that solution that has the largest or smallest objective-function value. This chapter demonstrates such a case for a large-scale, network-interdiction problem.

3.3 Background

To simplify the subsequent description of our explicit-enumeration method for solving Benders master problems, this section presents a standard BDA to solve a MIP. While Benders decomposition applies to MIPs with general integer variables, we limit ourselves to the common case with 0-1 variables only (e.g., Sa, 1969; Rana and Vickson, 1988; Sarin and West-Hansen, 2005; Brown et al., 2006).

3.3.1 Binary Mixed Integer Program

We wish to solve the following problem, assumed to have a finite optimal solution:

\[
\text{MIP}(\mathcal{X}, \mathcal{Y}) \quad z^* = \min_{x \in \mathcal{X}, y \in \mathcal{Y}} cx + fy \\
s.t. \quad Bx + Dy \geq d,
\]

(3.1)

(3.2)

where \(\mathcal{X} \equiv \{x \in \{0,1\}^{n_x} | Ax \leq b\} ; \mathcal{Y} \equiv \{y \in \mathbb{R}^{n_y}_+\} ; A, B, \) and \(D\) are dimensioned \(m_1 \times n_x, m_2 \times n_x, \) and \(m_2 \times n_y, \) respectively; the vectors \(b, c, d, f, x, \) and \(y\) conform; and, for ease
of exposition and without loss of generality, we assume that \( c = 0 \). In addition, we require that \( \mathcal{X} \) be easy to enumerate and have modest cardinality. Our test problem defines \( \mathcal{X} \) through a single knapsack constraint (see also Brown et al. 2006, and Sanchez and Wood 2006), so the first requirement is satisfied, and we shall see empirically that the second is also. More complex constraint sets defining \( \mathcal{X} \) are certainly possible, and we suggest those used in Geoffrion and Graves (1974) and Sarin and West-Hansen (2005) as good examples.

Fixing \( x = \hat{x} \) in MIP yields the following subproblem:

\[
\text{SUB}(\hat{x}, \mathcal{Y}) \quad z^*(\hat{x}) = \min_{y \in \mathcal{Y}} fy \\
\text{s.t. } Dy \geq d - B\hat{x}, \quad [\alpha(\hat{x})]
\]

where optimal dual variables are shown in brackets next to their corresponding constraints, and where \( y(\hat{x}) \) denotes the optimal primal variables. For later use, let \( \mathcal{A} \) denote the set of all extreme-point vectors for \( \{ \alpha | \alpha D \leq f, \alpha \geq 0 \} \). Without loss of generality, we assume that \( \alpha(\hat{x}) \) is unique and, for ease of exposition, we let \( \hat{\alpha} = \alpha(\hat{x}) \) and \( \hat{y} = y(\hat{x}) \).

**Assumption 3.1** SUB(\( \hat{x}, \mathcal{Y} \)) is feasible for any \( \hat{x} \in \mathcal{X} \).

Assumption 3.1 corresponds to “relatively complete recourse,” a commonly assumed property in the stochastic-programming literature (e.g., Wets, 1974). While convenient for exposition, the assumption is not limiting. If the assumption does not hold, then BDA can (i) introduce “feasibility cuts” to the master problem as necessary (Benders, 1962), or (ii) solve a reformulated problem that allows penalized constraint violations in the subproblem (Yuen et al., 2006).

### 3.3.2 Benders Decomposition Algorithm (BDA)

Under Assumption 3.1, Benders decomposition reformulates MIP(\( \mathcal{X}, \mathcal{Y} \)) into this equivalent master problem:
$$\text{MP}(\mathcal{X}, \mathcal{A}) \quad z^* = \min_{x \in \mathcal{X}, \eta} \eta$$

\[ \text{s.t. } \eta \geq \hat{\alpha}(d - Bx) \quad \forall \hat{\alpha} \in \mathcal{A}. \]  

(3.5) 

(3.6)

Rather than exhaustively computing the elements of \( \mathcal{A} \) and then solving \( \text{MP}(\mathcal{X}, \mathcal{A}) \) directly, however, a general version of BDA begins with an arbitrary set \( \hat{\mathcal{A}} \subset \mathcal{A} \), and with an initial feasible solution \( \hat{x} \in \mathcal{X} \). Then, iteratively, BDA (i) solves \( \text{SUB}(\hat{x}, \mathcal{Y}) \) for \( \hat{y} \) and \( \hat{\alpha} \), (ii) adds \( \hat{\alpha} \) to \( \hat{\mathcal{A}} \), and (iii) solves \( \text{MP}(\mathcal{X}, \hat{\mathcal{A}}) \) for \( \hat{x} \). The algorithm terminates once \( \text{MP}(\mathcal{X}, \hat{\mathcal{A}}) \) approximates \( \text{MP}(\mathcal{X}, \mathcal{A}) \) sufficiently to prove near-optimality of some previously discovered solution \( (\hat{x}, \hat{y}) \) (Benders, 1962). The complete algorithm is provided in Figure 3.1 for reference. Note that the algorithm contains both a subscript \( K \) and a dummy step to simplify subsequent descriptions.

Figure 3.1: Benders decomposition algorithm (BDA).

---

**Input:** An instance of \( \text{MIP}(\mathcal{X}, \mathcal{Y}) \) and allowable optimality gap \( \varepsilon \geq 0 \).

**Output:** An \( \varepsilon \)-optimal solution to \( \text{MIP}(\mathcal{X}, \mathcal{Y}) \).

**Note:** \( \bar{z} \) and \( \bar{\bar{z}} \) are lower and upper bounds on \( z^* \), respectively. \( \hat{\mathcal{A}}_K \) denotes the \( K \)th set of dual extreme-point vectors.

**Initialization**
1: Decompose \( \text{MIP}(\mathcal{X}, \mathcal{Y}) \) to \( \text{MP}(\mathcal{X}, \mathcal{A}) \) and \( \text{SUB}(\hat{x}, \mathcal{Y}) \).
2: \( K \leftarrow 0; \bar{z} \leftarrow -\infty; \bar{\bar{z}} \leftarrow \infty; \hat{\mathcal{A}}_0 \leftarrow \emptyset; \hat{x}_1 \leftarrow \text{any } x \in \mathcal{X} \);
3: Dummy step;

**Subproblem**
4: \( K \leftarrow K + 1 \);
5: Solve \( \text{SUB}(\hat{x}_K, \mathcal{Y}) \) for \( \hat{y}, \hat{\alpha}_K \), and objective value \( z^*(\hat{x}_K) \);
6: \( \hat{\mathcal{A}}_K \leftarrow \hat{\mathcal{A}}_{K-1} \cup \{\hat{\alpha}_K\} \);
7: if \( (z^*(\hat{x}_K) < \bar{z}) \) \{ \( \bar{z} \leftarrow z^*(\hat{x}_K); \) \( (x^*, y^*) \leftarrow (\hat{x}_K, \hat{y}) \) \};
8: if \( (\bar{\bar{z}} - \bar{z} \leq \varepsilon \bar{\bar{z}}) \) \{ Go to Step 12; \} /* We assume \( \bar{\bar{z}}, \bar{z} \geq 0 */

**Master Problem**
9: Solve \( \text{MP}(\mathcal{X}, \hat{\mathcal{A}}_K) \) for \( \hat{x}_{K+1} \) and objective value \( z^*_K \); \( \bar{z} \leftarrow z^*_K \);
10: if \( (\bar{\bar{z}} - \bar{z} \leq \varepsilon \bar{\bar{z}}) \) \{ Go to Step 12; \}
11: Go to Step 4;

**Print Solution**
12: Print ("\( \varepsilon \)-optimal solution to \( \text{MIP}(\mathcal{X}, \mathcal{Y}) \) is", \( (\bar{z}, x^*, y^*) \)); Stop.
Proposition 3.1 The Benders Decomposition Algorithm solves MIP(\(\mathcal{X}, \mathcal{Y}\)).

Proof. This follows from Benders’ original work (Benders, 1962). □

3.4 Master Problem Solution by Enumeration

Typically, BDA solves MP(\(\mathcal{X}, \hat{A}_K\)) by B&B, but if \(|\mathcal{X}|\) is sufficiently small, we will demonstrate empirically that explicit enumeration can solve MP(\(\mathcal{X}, \hat{A}_K\)) more efficiently. The comparative efficiency of enumeration will become apparent as the number of cuts in MP(\(\mathcal{X}, \hat{A}_K\)) increases with increasing \(K\), and B&B slows down dramatically.

3.4.1 Benders Decomposition Algorithm Using Enumeration to Solve Master Problems (BDEA)

Let \(\hat{\alpha}_k\) denote the dual vector generated in the \(k\)th iteration of BDA. The standard Benders master problem in iteration \(K\) is

\[
\text{MP}(\mathcal{X}, \hat{A}_K) \quad z^*_K = \min_{x \in \mathcal{X}, \eta} \eta \\
\text{s.t.} \eta \geq g_0(\hat{x}_k) + \sum_{j=1}^{n_x} g_j(\hat{x}_k)x_j \quad \forall \quad k \in \{1, \ldots, K\},
\]

where, letting \(B_{.j}\) denote \(B\)'s \(j\)th column, \(g_j(\hat{x}_k) \equiv -\hat{\alpha}_kB_{.j}\) for \(j = 1, \ldots, n_x\), and \(g_0(\hat{x}_k) \equiv \hat{\alpha}_kd\).

Equivalently, MP(\(\mathcal{X}, \hat{A}_K\)) may be written as

\[
\text{MP}(\mathcal{X}, \hat{A}_K) \quad z^*_K = \min_{x \in \mathcal{X}} \eta_K(x), \quad (3.9)
\]

where \(\eta_K(x) \equiv \max_{k=1, \ldots, K} g_0(\hat{x}_k) + \sum_{j=1}^{n_x} g_j(\hat{x}_k)x_j, \quad (3.10)
\]

and where \(\hat{x}_{K+1} \equiv \arg\min_{x \in \mathcal{X}} \{\eta_K(x)\}\).

Now, rather than solving MP(\(\mathcal{X}, \hat{A}_K\)) “from scratch” in each iteration as a mathematical program defined by (3.7) and (3.8), equations (3.9) and (3.10) lead us to a simple “update computation” for evaluating \(\eta_K(x)\) and thereby solving MP(\(\mathcal{X}, \hat{A}_K\)):
\[
\eta_K(x) \leftarrow \max \left\{ \eta_{K-1}(x), \ g_0(\hat{x}_K) + \sum_{j=1}^{n_x} g_j(\hat{x}_K)x_j \right\} \quad \forall \ x \in \mathcal{X}.
\]  

(3.11)

By storing \(\eta_{K-1}(x)\) for each \(x \in \mathcal{X}\), the computational effort to solve \(\text{MP}(\mathcal{X}, \hat{A}_K)\) will not increase with \(K\), as it tends to with standard B&B. (In effect, the new method saves all of the leaf nodes in a weakly pruned B&B tree and, thus, it may be viewed as a simplified version of the B&B restart mechanism described in Tuy and Horst 1988.) Figure 3.2 presents the modifications of Figure 3.1 that convert standard BDA into “BDEA,” a \textit{Benders decomposition algorithm that uses enumeration to solve the master problems}. BDEA is simply one particular implementation of BDA, so Proposition 3.1 still applies.

We note that BDEA also applies to two variants of Benders decomposition, \textit{generalized Benders decomposition} (“GBD”; see Geoffrion, 1972), and \textit{global Benders decomposition} (“GLBD”; see Salmeron et al., 2009). The theory of GBD (Bloom, 1983) invokes convexity and tells us that BDA will converge as long as each function \(g_0(\hat{x}) + \sum_{j=1}^{n_x} g_j(\hat{x})x_j\) defines a subgradient approximation of \(z^*(x)\) at \(\hat{x}\). (The function \(z^*(x)\) may include a convex nonlinear term \(f(x)\), also.) By contrast, the theory of GLBD simply recognizes that BDA, suitably modified, must converge for binary \(x\) if (i) \(g_0(\hat{x}) \equiv z^*(\hat{x})\) for all \(\hat{x} \in \mathcal{X}\), (ii) \(g_j(\hat{x}) \equiv 0\) for all \(j\) such that \(\hat{x}_j = 1\), and (iii) for each \(j\) such that \(\hat{x}_j = 0\), a cut coefficient \(g_j(\hat{x})\) can be computed such that \(z^*(x) \geq g_0(\hat{x}) + \sum_{j=1}^{n_x} g_j(\hat{x})x_j\) for all \(x, \hat{x} \in \mathcal{X}\). (More general forms of GLBD cuts may be possible, but this is the form used in Salmeron et al. 2009.) In fact, the

Figure 3.2: BDA using enumeration to solve master problems (BDEA).

---

Proceed as in BDA, Algorithm Figure 3.1, but with the following variations:
Replace Step 3 with: \(\eta_0(x) \leftarrow -\infty \quad \forall \ x \in \mathcal{X}\);
Replace Step 9 with:
\[
\text{for (all } x \in \mathcal{X} \text{) } \eta_K(x) \leftarrow \max \left\{ \eta_{K-1}(x), \ g_0(\hat{x}_K) + \sum_{j=1}^{n_x} g_j(\hat{x}_K)x_j \right\} ; \\
\hat{z}_K \leftarrow \min_{x \in \mathcal{X}} \{ \eta_K(x) \} ; \ z \leftarrow \hat{z}_K ; \\
\hat{x}_{K+1} \leftarrow \arg\min_{x \in \mathcal{X}} \{ \eta_K(x) \} ;
\]
computational example in the next section derives from GLBD.

3.4.2 Knapsack Enumeration Algorithm (KEA)

To solve \( \text{MP}(\mathcal{X}, \hat{A}) \) as described above, BDEA must generate every \( x \in \mathcal{X} \). Let us assume that \( \mathcal{X} \) is defined through a single knapsack constraint

\[
\mathcal{X} \equiv \{ x \in \{0,1\}^{n_x} \mid ax \leq b \}, \tag{3.12}
\]

where all data are positive integers. Horowitz and Sahni (1978, pp. 350-354) describe a backtracking knapsack algorithm (BKNAP1) for solving the following knapsack problem:

\[
\text{KNAP} \quad z_{\text{KNAP}}^* = \max cx \quad \text{s.t. } ax \leq b \quad x \in \{0,1\}^{n_x}, \tag{3.13}
\]

where \( a \) and \( c \) are \( n_x \)-vectors; \( a, c \geq 0 \); and \( b \geq 0 \).

We simply wish to enumerate all \( x \) in \( \mathcal{X} \) as we evaluate \( \eta_K(x) \). Therefore, we modify the enumeration steps in BKNAP1 to produce the Knapsack Enumeration Algorithm (KEA). KEA enumerates \( q \) feasible knapsack solutions \( x \) in reverse lexicographic order, beginning with the solution \( x^0 \). Now, if we are enumerating on a single processor, then \( q = |\mathcal{X}| \) and \( x^0 \) is the first element in a reverse lexicographic ordering of \( \mathcal{X} \). But, for parallel enumeration (see Section 3.4.3), \( q \) will be the cardinality of a particular subset of \( \mathcal{X} \), and \( x^0 \) will be the first element in a reverse lexicographic ordering of that subset’s elements.

In their implementation, Horowitz and Sahni store \( x \) using an \( n_x \)-element array. This results in repeated examination of zero-valued binary variables. The more efficient KEA uses a stack to store only the indices \( j \) such that \( x_j = 1 \).

We make two additional modifications to KEA for optimizing the master problems of Section 3.5. First, \( a_j = 1 \) for all \( j \in \{1, \ldots, n_x\} \), which simplifies KEA’s feasibility testing. Second, there exists an optimal solution to \( \text{MP}(\mathcal{X}, \hat{A}) \) with \( x^* \in \bar{\mathcal{X}} \), where \( x \in \bar{\mathcal{X}} \subset \mathcal{X} \) if and only if \( x \) is maximal for the knapsack constraint in (3.12). Thus, we only accept master-
problem solutions \( \mathbf{x} \in \tilde{\mathcal{X}} \) at Step 9 of BDEA, and modify KEA accordingly. Figure 3.3 provides the complete algorithm. The algorithm is easy to adjust if \( a \neq 1 \) or if \( \mathbf{x}^* \) is not necessarily maximal for the knapsack constraint.

### 3.4.3 Parallel Processing

Parallel processing can reduce the time required to solve a master problem by explicit enumeration. To solve \( \text{MP}(\mathcal{X}, \hat{A}_K) \) on \( P \) parallel processors, we first use KEA to enumerate \( \tilde{\mathcal{X}} \) and evaluate its cardinality. Next, we implicitly partition the solutions into ordered subsets \( \tilde{\mathcal{X}}_1, \tilde{\mathcal{X}}_2, \ldots, \tilde{\mathcal{X}}_P \), such that (i) each subset’s elements are in decreasing lexicographic order, (ii) \( |\tilde{\mathcal{X}}_p| \approx |\tilde{\mathcal{X}}|/P \) for all \( p \in \{1, \ldots, P\} \), and (iii) for \( p < p' \), \( \mathbf{x} \in \tilde{\mathcal{X}}_p \), and \( \mathbf{x}' \in \tilde{\mathcal{X}}_{p'} \),

Figure 3.3: Knapsack enumeration algorithm, modified from Horowitz and Sahni (1978).

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( i \leftarrow 0; \ cnt \leftarrow 0; \ S \leftarrow \text{empty stack} )</td>
</tr>
<tr>
<td>2</td>
<td>for (all ( j \in {1, \ldots, n_x} ))</td>
</tr>
<tr>
<td>3</td>
<td>if ( x_0^j = 1 ) { push ( j ) on ( S ); ( \ cnt \leftarrow \ cnt + 1 ); }</td>
</tr>
<tr>
<td>4</td>
<td>Go to Step 12;</td>
</tr>
<tr>
<td>5</td>
<td>if ( (i = q) ) { Stop. } /* We have enumerated ( q ) solutions. */</td>
</tr>
<tr>
<td>6</td>
<td>if ( (cnt = 0) ) { Stop. } /* Fewer than ( q ) feasible, maximal solutions available. */</td>
</tr>
<tr>
<td>7</td>
<td>Pop ( j ) from ( S );</td>
</tr>
<tr>
<td>8</td>
<td>Push ( j + 1 ) on ( S );</td>
</tr>
<tr>
<td>9</td>
<td>j \leftarrow \text{top of} \ S;</td>
</tr>
<tr>
<td>10</td>
<td>if ( (j &gt; n_x - (b - cnt)) ) { Pop ( j ) from ( S ); ( cnt \leftarrow cnt - 1 ); go to Step 5; }</td>
</tr>
<tr>
<td>11</td>
<td>while ( (cnt &lt; b) ) { j \leftarrow \text{top of} \ S; \ push ( j + 1 ) on ( S ); ( cnt \leftarrow cnt + 1 ); }</td>
</tr>
<tr>
<td>12</td>
<td>Print Solution</td>
</tr>
<tr>
<td>13</td>
<td>print Solution ( i ) is ”, ( S );</td>
</tr>
<tr>
<td>14</td>
<td>Go to Step 5;</td>
</tr>
</tbody>
</table>
we require that \( \mathbf{x} >_L \mathbf{x'} \), where \( >_L \) indicates the lexicographic inequality. This partitioning scheme permits KEA to regenerate an entire subset while only storing the subset’s first element and the subset’s cardinality.

For each master problem, a parallel version of BDEA can then (i) use a separate processor labeled \( p \) to evaluate \( \eta_K(\mathbf{x}) \) for each \( \mathbf{x} \in \tilde{X}^p \); (ii) let \( z^p_K \leftarrow \min_{\mathbf{x} \in \tilde{X}^p} \{ \eta_K(\mathbf{x}) \} \) and \( \hat{x}_{K+1}^p \leftarrow \arg\min_{\mathbf{x} \in \tilde{X}^p} \{ \eta_K(\mathbf{x}) \} \) for each \( p \in \{1, \ldots, P\} \); and finally (iii) let \( z^*_{K} \leftarrow \min_{p \in \{1, \ldots, P\}} \{ z^p_K \} \) and \( \hat{x}_{K+1} \leftarrow \arg\min_{p \in \{1, \ldots, P\}} \{ \eta_K(\hat{x}_{K+1}^p) \} \).

### 3.4.4 Multicut Master Problem

Extensions of Benders decomposition commonly solve stochastic programs, (e.g., Infanger, 1992; Laporte and Louveaux, 1993). Birge and Louveaux (1988) describe one such extension, a multicut BDA that exploits the independence of a stochastic program’s scenario subproblems. Rather than summing cut information from all of the subproblems to obtain a single master-problem cut (as in Van Slyke and Wets 1969), a multicut BDA generates up to one Benders cut per subproblem. Thus, \( \eta_K(\mathbf{x}) \) is replaced by \( \eta^1_K(\mathbf{x}), \ldots, \eta^{|S|}_K(\mathbf{x}) \), where \( S \) is a set of independent subproblems.

While we do not test a multicut version of BDEA, one is certainly implementable. To solve the multicut master problem, BDEA could (i) let \( \eta^s_K(\mathbf{x}) \leftarrow \max \left\{ \eta^{s-1}_K(\mathbf{x}), g_0^s(\tilde{x}_K) + \sum_{j=1}^{n_x} g_j^s(\tilde{x}_K) x_j \right\} \) for each \( \mathbf{x} \in \tilde{X} \) and \( s \in S \); (ii) let \( \eta_K(\mathbf{x}) \leftarrow \sum_{s=1}^{|S|} p^s \eta^s_K(\mathbf{x}) \) for each \( \mathbf{x} \in \tilde{X} \), where \( p^s \) is the probability, or weight, of subproblem \( s \); and finally (iii) let \( z^*_{K} \leftarrow \min_{\mathbf{x} \in \tilde{X}} \{ \eta_K(\mathbf{x}) \} \) and \( \hat{x}_{K+1} \leftarrow \arg\min_{\mathbf{x} \in \tilde{X}} \{ \eta_K(\mathbf{x}) \} \).

### 3.5 Computational Results

We test the explicit- enumeration method by solving the sequence of 1400 master problems that GLBD generates in Salmeron et al. (2009) as it solves a network-interdiction problem, denoted \( NI \) here. Rather than implement a complete BDEA, we simply solve \( \text{MP}(\mathcal{X}, \hat{A}_K) \) for \( K = 1, \ldots, 1400 \).
In \( N\), an attacker seeks to maximize the estimated cost of operating an electric power transmission grid by disabling up to five of the grid’s 113 substations. Converting the master problem in Salmeron et al. to a minimization, the resulting Benders cuts have the form of (3.8), where (i) \( x_j = 1 \) if the attacker disables substation \( j \), and \( x_j = 0 \), otherwise; (ii) \(-g_0(\hat{x})\) represents the estimated network operating cost after the attacker executes interdiction plan \( \hat{x} \); (iii) if \( \hat{x}_j = 0 \), then \(-g_j(\hat{x})\) represents an upper bound on the additional cost that would result from disabling substation \( j \); and (iv) if \( \hat{x}_j = 1 \), then \( g_j(\hat{x}) = 0 \). Since disabling a substation cannot decrease overall operating cost, an optimal attack plan will be maximal with respect to “attack resource” and, therefore, will disable exactly five substations. Thus, BDEA can solve \( N\) by replacing the minimization over \( \mathcal{X} \) in (3.7), and elsewhere, with a minimization over \( \mathcal{X}' \).

We implement explicit enumeration in C++, using the GNU G++ compiler, OpenMP parallelization (OpenMP Architecture Review Board, 2014), and up to eight Intel Xeon E5520 quad-core processors on a computer with 12 GB of memory. For comparison, we obtain B&B solution times on the same machine using CPLEX solver 12.4 with default parameter settings except: the optimality gap is relaxed to 0.1%; processing times are displayed; deterministic parallelism is used; and the number of parallel processors \( P \) is specified.

### 3.5.1 Basic Results

Computational tests here compare the time required for explicit enumeration and B&B to solve the entire series of 1400 master problems from Salmeron et al. (2009). In reality, BDEA and BDA might produce distinct solution trajectories, with BDEA’s exact master-problem solutions and BDA’s solutions with some non-zero optimality tolerance yielding different iteration counts; see Salmerón and Wood (2014). But, if explicit enumeration solves this sequence of master problems substantially faster than B&B, then BDEA likely solves \( N\) faster than a standard BDA.

For both explicit enumeration and B&B, using either one or eight processors, Figure 3.4 illustrates the total seconds required to solve 100\( q \) master problems, for \( q = 0, \ldots, 14 \). Note
Figure 3.4: Comparing methods for solving Benders master problems. (a) The plot depicts the total master problem solution time (in seconds) for a specified number of Benders iterations. Note that the solution times are shown on a logarithmic scale. (b) This plot truncates the vertical scale to emphasize that total run time for explicit enumeration is essentially linear in the number of master problems solved, $K$. (The initial enumeration of $\bar{X}$, the set of maximal solutions, creates an offset at $K = 0$ as shown on the logarithmic scale, but that offset is clearly negligible here.)
that \( q = 14 \) implies “all master problems,” and that the apparently negligible explicit-enumeration time for \( q = 0 \) is the time required to enumerate \( \hat{X} \).

The results in Figure 3.4 can be summarized as follows. With a single processor, explicit enumeration reduces total solution time by a factor of almost 8 over B&B; with eight processors that factor is roughly 18. Not shown are results for \( P = 2, 4, \) and 6 processors. For these cases, parallel enumeration yields total solution times of 1978, 1034, and 575 seconds, respectively, while parallel B&B yields total solution times of 17469, 11713, and 9513 seconds, respectively.

Naturally, the solution time per iteration for explicit enumeration remains essentially constant: on average, 2.15 seconds with one processor and 0.34 seconds with eight. By contrast, solution times increase dramatically for B&B as the size of \( \text{MP}(\mathcal{X}, \hat{A}_K) \) increases. For example, using one processor, each of the first 100 master problems requires an average of 0.52 seconds to solve, while each of the final 100 requires 64.32 seconds; with eight processors, the respective times are 0.52 and 20.41 seconds.

### 3.5.2 Lazy Constraints

For completeness, we also explore how using lazy constraints might reduce the time required for BDA to solve \( \text{MIP}(\mathcal{X}, \mathcal{Y}) \). For any particular master problem \( \text{MP}(\mathcal{X}, \hat{A}_K) \), many of the Benders cuts are “loose,” i.e.,

\[
 z^*_K \gg g_0(\hat{x}_k) + \sum_{j=1}^{n_x} g_j(\hat{x}_k)x^*_j \tag{3.16}
\]

for numerous \( k \in \{1, ..., K\} \), where \( x^* \) is an \( \varepsilon \)-optimal solution to \( \text{MP}(\mathcal{X}, \hat{A}_K) \). Unfortunately, loose cuts can only be definitively identified after solving \( \text{MP}(\mathcal{X}, \hat{A}_K) \). We can guess that a cut is more likely to be loose in iteration \( K \), however, if it is either (i) “old,” that is, from an iteration \( k \) where \( k \ll K \), or (ii) loose in iteration \( K - 1 \).

For \( \kappa \in \{400, 900, 1300, 1325, 1350\} \) and using both one and eight processors, we solve the final 30 master problems from the sequence of 1400 using B&B while letting the \( \kappa \) “oldest”
Table 3.1: Solution times for B&B solving the final 30 (of 1400) master problems while letting either the oldest or the loosest cuts be lazy.

<table>
<thead>
<tr>
<th>Processors (number)</th>
<th>Total solution time with no lazy cuts (sec.)</th>
<th>Total solution time with lazy cuts (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Number of oldest cuts treated as lazy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>400 900 1300 1325 1350</td>
</tr>
<tr>
<td>1</td>
<td>1958</td>
<td>2043 1927 1819 1769 1776</td>
</tr>
<tr>
<td>8</td>
<td>609</td>
<td>716 675 654 650 642</td>
</tr>
</tbody>
</table>

cuts, i.e., \( k \in \{1, \ldots, \kappa\} \), be lazy. (While this is a small subset of the 1400 master problems, it contains the largest master problems, which presumably have the largest numbers of loose cuts.) We next solve the same 30 master problems while letting the \( \kappa \) “loosest” cuts from iteration \( K - 1 \) be lazy in iteration \( K \). Table 3.1 shows the results, which include B&B solution times without applying lazy cuts for comparison. The results indicate that lazy constraints may enable BDA to solve \( \text{MIP}(\mathcal{X}, \mathcal{Y}) \) more quickly. However, the improvement is insignificant relative to the speedup produced by applying BDEA.

3.6 Conclusions

Because master-problem solutions consume the bulk of the time that GLBD uses to solve \( NI \) in Salmeron et al. (2009), it is clear that BDEA would solve the full model much faster than the standard implementation used in that paper as long as the total number of iterations is comparable. We are currently working to implement a complete BDEA algorithm for a MIP, including a multicut version with a stochastic-programming application in mind.
4.1 Abstract

We solve two-stage stochastic mixed-integer programs using an implementation of Benders decomposition that solves master problems by enumerating all feasible solutions. For both “single-cut” and multicut variants, storing objective-function values for all master-problem solutions ensures that the computational effort required to optimize that objective function (by enumeration) remains constant from one iteration to the next, rather than increasing as in a typical Benders implementation. A stochastic capacitated facility-location problem illustrates. Without solving subproblems in parallel, an enumerative single-cut algorithm reduces average solution times for a 128-scenario instance from $T$ to as low as $0.26T$. By also solving subproblems in parallel on eight processors, we estimate that the enumerative algorithm could reduce solution times to as little as $0.04T$. For a 1024-scenario problem, we estimate that a 1024-processor implementation of the enumerative algorithm could provide a 97% reduction in solution time compared to a single-processor implementation.

4.2 Introduction

Benders decomposition (Benders, 1962) is an important tool for solving mixed-integer programs (MIPs). Applications include train locomotive and car assignment (Cordeau et al., 2000), mine production scheduling (Sarin and West-Hansen, 2005), and power-system operation (Van Dinter et al., 2013). The $L$-shaped algorithm of Van Slyke and Wets (1969) adapts Benders decomposition to solve two-stage stochastic linear programs; multiple authors (e.g.,
Cormican et al., 1998; MirHassani et al., 2000) extend this algorithm to two-stage stochastic mixed-integer programs (2SSPs) with integer variables in the first stage. The current chapter shows how to solve 2SSPs using an implementation of Benders decomposition that optimizes the master-problem objective function by explicit enumeration, with the goal of reducing overall solution times substantially.

The Benders decomposition algorithm (BDA) alternates between solving (i) a mixed-integer master problem, which optimizes a piecewise-linear, cutting-plane approximation of 2SSP’s objective function, and (ii) a set of subproblems, which generates one or more master-problem cuts during each iteration. (As in the preceding chapter, the current chapter does not consider “branch-and-cut” variants of BDA, e.g., Wu 2013.) The cuts improve the approximation, but each cut tends to make the mixed-integer master problem more difficult to solve (Naoum-Sawaya and Elhedhli, 2013). As Holmberg (1994) notes, the master problems can become difficult enough that BDA becomes too slow for practical use; see also McDaniel and Devine (1977) and Rana and Vickson (1988). (In fact, “too slow” occurs in our testing.)

Typical BDA implementations solve the master problem using linear-programming-based branch-and-bound (B&B). But, for a common class of problems whose members require a sufficiently large number of Benders iterations yet have relatively few feasible solutions to the master-problem—fewer than $10^9$, say—we will show how to reduce overall solution times by solving the master problem using explicit enumeration instead.

Salmerón and Wood (2014) implement a “Global Benders Decomposition” algorithm that solves master problems by explicit enumeration. Initial results are promising, although they are somewhat inconclusive because these authors program the enumeration using a slow algebraic modeling language and do not employ any parallel processing. In Chapter 3, we have demonstrated the potential of a “Benders decomposition algorithm using enumeration to solve master problems” (BDEA) by comparing the time required for either enumeration or B&B to solve a sequence of master problems from Salmeron et al. (2009). While enumeration
provides an order-of-magnitude speedup of master-problem solutions, Chapter 3 does not implement a complete Benders algorithm. That is, instead of solving subproblems to generate Benders cuts dynamically, that chapter solved a series of master problems whose cuts were given in advance. The current chapter provides full BDA and BDEA implementations for both the “single-cut” Benders master problem as in Van Slyke and Wets (1969), and also the “multicut” master problem of Birge and Louveaux (1988). We demonstrate BDEA’s performance on the equivalent deterministic formulation of a two-stage stochastic program (Infanger, 1992). Our particular two-stage stochastic facility-location problem has up to 1024 scenarios: as monolithic MIPs, these models can have one-million variables and seventy-thousand constraints. Rapid solution of such a deterministic equivalent also means that BDEA will be useful for solving the approximating problems generated by sampling the stochastic parameters of more general 2SSPs (e.g., Higle and Sen, 1999; Mak et al., 1999).

If fixing the master-problem variables yields multiple independent subproblems, then BDA or BDEA can solve those subproblems in parallel (Geoffrion, 1972; Dantzig and Glynn, 1990; Vladimirov and Zenios, 1999). BDA can also solve master problems using parallel B&B (as our implementations in Section 4.6 do). A key advantage for BDEA, however, is that explicit parallel processing can reduce master-problem solution times. For example, rather than solve the master problem by computing the objective-function value for every feasible master-problem solution on a single processor, single-cut BDEA can partition the feasible master-problem solutions into subsets and compute objective-function values for each subset on a separate processor. (See the preceding chapter.) We also show how BDEA can solve multicut master problems in parallel with the eight processors available to us.

Benders decomposition may prove inappropriate for certain MIPs, for instance those that are easy to solve or those that fail to yield “easy” subproblems after fixing integer variables. We believe, however, that BDEA can help solve 2SSPs and other problems in the literature, including examples from system interdiction (Brown et al., 2006; Lim and Smith, 2007; Scaparra and Church, 2008), network design (Melkote and Daskin, 2001; Randazzo
and Luna, 2001), and facility location (Rolland et al., 1997; Alp et al., 2003). Although there may be many binary variables in each of these instances (one for every possible attack on a target, connection between two network nodes, or potential facility sites, respectively), only a few of those binary variables will take on the value 1 in any optimal solution, and thus there are relatively few master-problem solutions.

The remainder of this chapter is organized as follows. Section 4.3 reviews algorithms for both single-cut and multicut Benders decompositions for a MIP with independent subproblems. Section 4.4 defines a BDEA for a single-cut decomposition and then extends the method to solve a multicut decomposition. Section 4.5 describes opportunities for parallel processing to reduce BDEA solution times. Section 4.6 provides computational results. Section 4.7 summarizes and provides recommendations for future research. Henceforth, we use the term “single-cut Benders decomposition algorithm” to refer to the L-shaped algorithm of Van Slyke and Wets (1969), modified to solve a 2SSP with integer variables in the first stage.

4.3 Preliminaries

BDEA extends BDA by applying an alternate master-problem solution technique. Therefore, to introduce notation and provide a reference point, we first present “single-cut BDA” (SCA) and “multicut BDA” (MCA) for solving a MIP with independent subproblems. We specialize to a 2SSP with scenarios indexed by $s$, with binary first-stage variables, and with a tractable deterministic equivalent formulation:

$$2SSP(\mathcal{X}, \mathcal{Y}, S) \quad z^* = \min_{\substack{x \in \{0,1\}^n_x \forall x \in \mathcal{X} \cap s \in S \forall s \in S}} \quad cx + \sum_{s \in S} p^s f y^s$$

s.t. $Bx + Dy^s \geq d^s \forall s \in S,$

where $\mathcal{X} \equiv \{x \in \{0,1\}^{n_x} \mid A x \leq b\}; \mathcal{Y} \equiv \{y \in \mathcal{R}^{n_y}_+\}; \mathcal{S}$ is a set of scenarios; $p^s$ is the probability that scenario $s \in \mathcal{S}$ occurs; $A$ and $B$ have dimensions $m_1 \times n_x$ and $m_2 \times n_x$, respectively; $D$ is dimensioned $m_2 \times n_y$; and all vectors conform. (All of the methods
developed here extend to more general problems with $B^s$, $D^s$, and $f^s$; for example, see Walkup and Wets 1967.) Let us assume that all $x \in \mathcal{X}$ can be enumerated easily; say $|\mathcal{X}|$ is at most $10^9$.

For $x = \hat{x} \in \mathcal{X}$, $2SSP(\mathcal{X}, \mathcal{Y}, \mathcal{S})$ yields the following subproblem for each $s \in \mathcal{S}$:

$$
\text{SUB}^s(\hat{x}, \mathcal{Y}) \quad z^s(\hat{x}) = \min_{y^s \in \mathcal{Y}} f y^s
$$

$$
\text{s.t. } D y^s \geq d^s - B \hat{x}, \quad [\alpha^s(\hat{x})]
$$

where $\alpha^s(\hat{x})$ denotes the optimal dual-variable values for constraints (4.4). Note that a Benders decomposition implementation does not depend upon, and we will not report, $y^s(\hat{x})$.

For ease of exposition, we let $\hat{\alpha}^s \equiv \alpha^s(\hat{x})$ and make the following assumption:

**Assumption 4.1** $2SSP(\mathcal{X}, \mathcal{Y}, \mathcal{S})$ exhibits the property of relatively complete recourse; that is, $\text{SUB}^s(\hat{x}, \mathcal{Y})$ is feasible for any combination of $\hat{x} \in \mathcal{X}$ and $s \in \mathcal{S}$.

Assumption 4.1 is common (e.g., Wets, 1974; Philpott and de Matos, 2012), and stochastic programs that have this property tend to possess special structures that may reduce computational effort compared to those that do not (Rockafellar and Wets, 1976). If $\text{SUB}^s(\hat{x}, \mathcal{Y})$ is not necessarily feasible, then SCA can either implement “feasibility cuts” (Benders, 1962), or permit penalized constraint violations in $\text{SUB}^s(\hat{x}, \mathcal{Y})$ (Yuen et al., 2006).

Under relatively complete recourse, the following Benders master problem is equivalent to $2SSP(\mathcal{X}, \mathcal{Y}, \mathcal{S})$:

$$
\text{MP}(\mathcal{X}, \mathcal{A}) \quad z^* = \min_{x \in \mathcal{X}, \eta} cx + \eta
$$

$$
\text{s.t. } \eta \geq \sum_{s \in \mathcal{S}} p^s \hat{\alpha}^s_k (d^s - Bx) \quad \forall \; k \in \{1, \ldots, |\mathcal{A}|\},
$$

where, for each $s \in \mathcal{S}$, $\mathcal{A}^s$ denotes the set of all extreme-point vectors for $\{\alpha^s|\alpha^s D \leq f, \alpha^s \geq 0\}$, and $\mathcal{A} \equiv \mathcal{A}^1 \times \ldots \times \mathcal{A}^{|\mathcal{S}|}$.

Given an initial feasible solution $\hat{x}_1 \in \mathcal{X}$ and an arbitrary subset $\hat{\mathcal{A}}_0 \subset \mathcal{A}$, SCA for $2SSP$ iteratively (i) solves $\text{SUB}^s(\hat{x}_K, \mathcal{Y})$ for each $s \in \mathcal{S}$ to obtain $(\hat{\alpha}^1_k, \ldots, \hat{\alpha}^{|\mathcal{S}|}_K)$; (ii) creates $\hat{\mathcal{A}}_K$
Figure 4.1: Single-cut Benders decomposition algorithm for 2SSP(\(X, Y, S\)) (SCA).

**Input:** An instance of 2SSP(\(X, Y, S\)) and allowable Benders “optimality gap” \(\varepsilon_{BD} \geq 0\).

**Output:** An \(\varepsilon_{BD}\)-optimal first-stage solution to 2SSP(\(X, Y, S\)).

**Note:** \(\bar{z}\) and \(\bar{z}\) are upper and lower bounds on \(z^*\), respectively. \(\hat{A}_K\) denotes the subset of \(A\) during iteration \(K\).

1: \(K \leftarrow 0; \bar{z} \leftarrow -\infty; \bar{z} \leftarrow \infty; \hat{x}_1 \leftarrow \text{any } x \in X; \hat{A}_0 \leftarrow \emptyset;\)

**Subproblems**

2: \(K \leftarrow K + 1;\)

3: for (all \(s \in S\)) solve \(\text{SUB}^s(\hat{x}_K, Y)\) for \(\hat{\alpha}_K^s\) and objective value \(\hat{z}^s(\hat{x}_K);\)

4: \(\hat{z}^*(\hat{x}_K) \leftarrow \sum_{s \in S} p^s z^s(\hat{x}_K);\)

5: \(\hat{A}_K \leftarrow \hat{A}_{K-1} \cup \{ (\hat{\alpha}_K^1, \ldots, \hat{\alpha}_K^{|S|}) \};\)

6: if \((\hat{z}^*(\hat{x}_K) < \bar{z})\) \{ \(\bar{z} \leftarrow \hat{z}^*(\hat{x}_K); x^* \leftarrow \hat{x}_K;\) \}

7: if \((\bar{z} - \bar{z} \leq \varepsilon_{BD} \cdot \bar{z})\) \{ Go to Step 11; \} /* We assume \(\bar{z}, \bar{z} \geq 0\) */

**Master Problem**

8: Solve \(\text{MP}(X, \hat{A}_K)\) for \(\hat{x}_{K+1}\) and objective value \(\hat{z}_K^*; \bar{z} \leftarrow \hat{z}_K^*;\)

9: if \((\bar{z} - \bar{z} \leq \varepsilon_{BD} \cdot \bar{z})\) \{ Go to Step 11; \}

10: Go to Step 2;

**Print Solution**

11: Print (“\(\varepsilon_{BD}\)-optimal first-stage solution to 2SSP(\(X, Y, S\)) is\(\{, (\bar{z}, x^*)\); Stop."

by adding \((\hat{\alpha}_K^1, \ldots, \hat{\alpha}_K^{|S|})\) to \(\hat{A}_{K-1}\); and (iii) solves \(\text{MP}(X, \hat{A}_K)\) for \(\hat{x}_{K+1}\). SCA terminates once \(\hat{A}_K\) suffices to prove near-optimality of some previous solution \(\hat{x}\) (Van Slyke and Wets, 1969). Figure 4.1 provides a standard SCA for 2SSP(\(X, Y, S\)). (This is a mixed-integer version of the \(L\)-shaped algorithm in Van Slyke and Wets 1969.)

**Proposition 4.1** The single-cut Benders decomposition algorithm (SCA) solves 2SSP(\(X, Y, S\)).

**Proof.** See Benders (1962) and Van Slyke and Wets (1969).

Rather than combine the vector elements of \((\hat{\alpha}_K^1, \ldots, \hat{\alpha}_K^{|S|})\) to create a single Benders cut in the \(K\)th iteration, Birge and Louveaux (1988) propose constructing one cut per subproblem. The multicut master problem is formulated as:
where, again, for each \( s \in S \), \( \mathcal{A}^s \) denotes the set of all extreme-point vectors for \( \{ \alpha^s | \alpha^s D \leq f, \alpha^s \geq 0 \} \).

After solving the subproblems, a standard multicut BDA (MCA) introduces \(|S|\) master-problem cuts of the form in (4.8), rather than a linear combination of those cuts as in (4.6). Intuitively speaking, each MCA iteration provides more information to the master problem than a corresponding SCA iteration does, so MCA tends to require fewer iterations to converge (Birge and Louveaux, 2011, p. 200). This reduction in iterations comes at the expense of increased master-problem size, however. Figure 4.2 presents MCA.

---

**Figure 4.2:** Multicut Benders decomposition algorithm for 2SSP(\(\mathcal{X}, \mathcal{Y}, S\)) (MCA).

---

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>(K \leftarrow 0); (\bar{z} \leftarrow -\infty); (\bar{\bar{z}} \leftarrow \infty); (\hat{x}_1 \leftarrow \text{any } x \in \mathcal{X}); (\hat{\mathcal{A}}_0^s \leftarrow \emptyset \forall s \in S);</td>
</tr>
<tr>
<td>2.</td>
<td>(K \leftarrow K + 1);</td>
</tr>
<tr>
<td>3.</td>
<td>for (all ( s \in S )) solve SUB(^<em>)((\hat{x}_K, \mathcal{Y})) for (\hat{\alpha}_K^s) and objective value (z^</em>(\hat{x}_K));</td>
</tr>
<tr>
<td>4.</td>
<td>(z^<em>(\hat{x}<em>K) \leftarrow \sum</em>{s \in S} p^s z^</em>(\hat{x}_K));</td>
</tr>
<tr>
<td>5.</td>
<td>(\hat{\mathcal{A}}<em>K^s \leftarrow \hat{\mathcal{A}}</em>{K-1}^s \cup { \hat{\alpha}_K^s } \forall s \in S);</td>
</tr>
<tr>
<td>6.</td>
<td>if ((z^<em>(\hat{x}_K) &lt; \bar{z})) { (\bar{z} \leftarrow z^</em>(\hat{x}_K)); (x^* \leftarrow \hat{x}_K); }</td>
</tr>
<tr>
<td>7.</td>
<td>if ((\bar{z} - \bar{\bar{z}} \leq \varepsilon_{BD} \cdot \bar{z})) { Go to Step 11; } /* We assume (\bar{z}, \bar{\bar{z}} \geq 0) */</td>
</tr>
<tr>
<td>8.</td>
<td>Solve MC-MP((\mathcal{X}, \hat{\mathcal{A}}<em>K)) for (\hat{x}</em>{K+1}) and objective value (z^<em>_K); (\hat{z} \leftarrow z^</em>_K);</td>
</tr>
<tr>
<td>9.</td>
<td>if ((\bar{z} - \bar{\bar{z}} \leq \varepsilon_{BD} \cdot \bar{z})) { Go to Step 11; }</td>
</tr>
<tr>
<td>10.</td>
<td>Go to Step 2;</td>
</tr>
<tr>
<td>11.</td>
<td>Print (&quot;(\varepsilon_{BD})-optimal first-stage solution to 2SSP((\mathcal{X}, \mathcal{Y}, S)) is(^\dagger), ((\bar{z}, x^*))); Stop.</td>
</tr>
</tbody>
</table>
Proposition 4.2. The multicut Benders decomposition algorithm (MCA) solves $2SSP(\mathcal{X}, \mathcal{Y}, S)$.


4.4 Master-Problem Solution by Enumeration

Implementations of SCA and MCA usually solve master problems by B&B. The decomposition of certain MIPs, however, can yield master problems that solve more efficiently by enumeration than by B&B (see Chapter 3). Enumeration will be viable if $|\mathcal{X}|$ is not “too large,” and it may be beneficial if the number of Benders iterations is large enough that the correspondingly large master problems become difficult to solve by B&B. In this section, we describe two BDEA implementations: (i) “single-cut BDEA” (SCEA), and (ii) “multicut BDEA” (MCEA).

4.4.1 Single-Cut Benders Decomposition Algorithm Using Enumeration to Solve Master Problems (SCEA)

The $K$th Benders master problem for SCA is:

$$\begin{align*}
\text{MP}(\mathcal{X}, \hat{A}_K) & \quad z^*_K = \min_{x \in \mathcal{X}, \eta} cx + \eta \\
& \quad \text{s.t. } \eta \geq \sum_{s \in S} p^s \hat{\alpha}^s_k(d^s - Bx) \forall k \in \{1, \ldots, K\}. 
\end{align*}$$

The computational effort required for B&B to solve $\text{MP}(\mathcal{X}, \hat{A}_K)$, as defined by (4.9) and (4.10), tends to increase with $K$ (Salmerón and Wood, 2014). But, $\text{MP}(\mathcal{X}, \hat{A}_K)$ may be expressed as:

$$\begin{align*}
\text{MP}(\mathcal{X}, \hat{A}_K) & \quad z^*_K = \min_{x \in \mathcal{X}} cx + \eta_K(x), \\
& \quad \text{where } \eta_K(x) = \max_{k \in \{1, \ldots, K\}} \sum_{s \in S} p^s \hat{\alpha}^s_k(d^s - Bx),
\end{align*}$$

and where the solution to $\text{MP}(\mathcal{X}, \hat{A}_K)$ is $\hat{x}_{K+1} = \arg\min_{x \in \mathcal{X}} \{cx + \eta_K(x)\}$. If we store $\eta_{K-1}(x)$ for each $x \in \mathcal{X}$, this formulation allows a simple “update” to calculate $\eta_K(x)$ for each $x \in \mathcal{X}$, and thus the computational effort required to solve $\text{MP}(\mathcal{X}, \hat{A}_K)$ remains constant as
$K$ increases. Figure 4.3 describes the conversion of standard SCA to SCEA. Except for its master-problem solution method, SCEA is identical to SCA, and thus Proposition 4.1 holds for SCEA as well as SCA.

### 4.4.2 Multicut Benders Decomposition Algorithm Using Enumeration to Solve Master Problems (MCEA)

The standard multicut master problem in iteration $K$ is:

$$
\text{MC-MP}(\mathcal{X}, \hat{A}_K) \quad z^*_K = \min_{x \in \mathcal{X}, \eta^s \forall s \in \mathcal{S}} \{c^T x + \sum_{s \in \mathcal{S}} p^s \eta^s \}
$$

(subject to $\eta^s \geq \hat{\alpha}_k^s (d^s - B^T x)$ for all $s \in \mathcal{S}$, $k \in \{1, \ldots, K\}$.)

Equivalently, the multicut master problem may be written:

$$
\text{MC-MP}(\mathcal{X}, \hat{A}_K) \quad z^*_K = \min_{x \in \mathcal{X}, \eta^s \forall s \in \mathcal{S}} \{c^T x + \sum_{s \in \mathcal{S}} p^s \eta^s_K(x) \}
$$

where $\eta^s_K(x) \equiv \max_{k \in \{1, \ldots, K\}} \hat{\alpha}_k^s(d^s - B^T x)$ for all $s \in \mathcal{S}$.

Again, $\hat{x}_{K+1}$ denotes the solution to this $K$th master problem. Figure 4.4 presents the modifications that convert MCA into MCEA. While an SCEA implementation must store the $|\mathcal{X}|$-element objective-function array $\eta_K(x)$, MCEA must store an $|\mathcal{X}|$-element subfunction

Figure 4.3: Single-cut BDA using enumeration to solve master problems for $2\text{SSP}(\mathcal{X}, \mathcal{Y}, \mathcal{S})$ (SCEA).

---

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replace Step 1 with:</td>
<td>$K \leftarrow 0; ; \hat{z} \leftarrow -\infty; ; \bar{z} \leftarrow \infty; ; \hat{x}_1 \leftarrow \text{any } x \in \mathcal{X}; ; \hat{A}_0 \leftarrow \emptyset; ; \eta_0(x) \leftarrow -\infty \forall x \in \mathcal{X};$</td>
</tr>
<tr>
<td>Replace Step 8 with:</td>
<td>8a: <strong>Update objective function:</strong> for (all $x \in \mathcal{X}$) $\eta_K(x) \leftarrow \max {\eta_{K-1}(x), \sum_{s \in \mathcal{S}} p^s \hat{\alpha}<em>K^s(d^s - B^T x)}$; 8b: <strong>Optimize approximation:</strong> $z^<em><em>K \leftarrow \min</em>{x \in \mathcal{X}} {c^T x + \eta_K(x)}; ; \hat{z} \leftarrow z^</em><em>K; ; \hat{x}</em>{K+1} \leftarrow \arg \min</em>{x \in \mathcal{X}} {c^T x + \eta_K(x)};$</td>
</tr>
</tbody>
</table>

53
array \eta_s^K(x) for each \( s \in S \). This requirement could make random-access-memory storage of these values impractical for certain 2SSPs. Of course, Proposition 4.2 holds for MCEA as well as MCA.

4.5 Opportunities for Parallel Computation

Both SCEA and MCEA provide opportunities for parallel processing to reduce solution times.

4.5.1 Single-cut Benders Decomposition Algorithm Using Enumeration to Solve Master Problems (SCEA)

Parallel processing could improve SCEA’s performance in two ways. SCEA, like SCA (e.g., Fragni`ere et al., 2000), can solve multiple subproblems in parallel during Step 3. But, unlike SCA, SCEA can also solve any master problem in parallel by (i) partitioning \( \mathcal{X} \) into subsets \( \mathcal{X}_1, \ldots, \mathcal{X}_N \); (ii) using a separate processor for each \( n \in \{1, \ldots, N\} \) to update the objective function \( \eta_K(x) \) for each \( x \in \mathcal{X}^n \) in Step 8a; and (iii) consolidating each processor’s results.

This thesis does not implement subproblem solutions in parallel as other authors have (e.g., Nielsen and Zenios, 1997; Dempster and Thompson, 1999). As long as BDEA and BDA

---

Figure 4.4: Multicut BDA using enumeration to solve master problems for 2SSP(\( \mathcal{X}, \mathcal{Y}, \mathcal{S} \)) (MCEA).

---

Proceed as in MCA, Figure 4.2, but with the following variations:

Replace Step 1 with: \( K \leftarrow 0; \; \tilde{z} \leftarrow -\infty; \; \tilde{z} \leftarrow \infty; \; \hat{\mathcal{A}}_0^s \leftarrow \emptyset \; \forall \; s \in \mathcal{S}; \; \hat{x}_1 \leftarrow \text{any} \; x \in \mathcal{X}; \)

\[ \eta_0^s(x) \leftarrow -\infty \; \forall \; x \in \mathcal{X}, \; s \in \mathcal{S}; \]

Replace Step 8 with:

8a: Update subfunctions:

\[ \text{for (all } x \in \mathcal{X}, \; s \in \mathcal{S}) \; \eta_s^K(x) \leftarrow \max \{ \eta_{K-1}^s(x), \; \hat{\alpha}_s^K(d^s - Bx) \}; \]

8b: Calculate objective function:

\[ \text{for (all } x \in \mathcal{X}) \; \eta_K(x) \leftarrow \sum_{s \in \mathcal{S}} p_s \eta_s^K(x); \]

8c: Optimize approximation:

\[ z_K^* \leftarrow \min_{x \in \mathcal{X}} \{ cx + \eta_K(x) \}; \; \tilde{z} \leftarrow z_K^*; \]

\[ \hat{x}_{K+1} \leftarrow \arg\min_{x \in \mathcal{X}} \{ cx + \eta_K(x) \}; \]
require approximately the same number of iterations, solving the subproblems in parallel would reduce overall solution times for each algorithm by approximately the same amount, and thus provide no relative benefit to either. Since $|\mathcal{X}|$ is only $6.17 \times 10^5$ for the test problems in Section 4.6, we do not bother to update $\eta_K(x)$ in parallel as in Chapter 3 (where $|\mathcal{X}| \approx 1.47 \times 10^8$). For a fair comparison, we do apply parallel B&B to the SCA master problems, however.

4.5.2 Multicut Benders Decomposition Algorithm Using Enumeration to Solve Master Problems (MCEA)

MCEA could use parallel processing to solve subproblems or to update subfunctions in the manner described in Section 4.5.1. (Our implementation does neither.) But, since the multicut decomposition yields $|S|$ individual cuts, MCEA can also update subfunction values $\eta^*_K(x)$ in parallel at Step 8a by (i) partitioning the $|S|$ subfunctions $\eta^*_K(x)$ into $N$ subsets $S^1, \ldots, S^N$; and (ii) using a separate processor for each $n \in \{1, \ldots, N\}$ to update subfunctions for all $s \in S^n$. If each subfunction update has the same execution time, then a parallel implementation of Step 8a has a theoretical efficiency $E = T_1/(N \cdot T_N) = 1$, where $T_1$ and $T_N$ are the serial and parallel execution times, respectively (Heroux et al., 2006, p. 2). We do implement this parallel MCEA method and compare against parallel B&B as part of the MCA implementation.

In addition to partitioning subfunctions and updating them on parallel processors, MCEA can also use parallel processing at Step 8b to calculate the objective-function value $\eta_K(x)$ for each $x$ in $\mathcal{X}$. While our implementation uses serial processing and two nested loops to calculate the $|\mathcal{X}|$ objective-function values, a parallel “binary-addition” method would require less time. Hillis and Steele Jr. (1986) compute the sum of $n$ numbers in parallel by creating a binary tree with the $n$ numbers as leaves and then using parallel processors to calculate the binary sums at each “branch” level in the tree. Similarly, binary addition can compute $\eta_K(x)$ in parallel by creating a binary tree with individual subfunction arrays as leaves, and then using parallel processing to combine arrays at each level. At the first branch
level, the binary-addition method would combine pairs of subfunction arrays that have the form of (4.16). The second level would combine pairs of paired arrays, the third level would combine pairs of pairs-of-paired arrays, and so on. Using $\log \equiv \log_2$, the last of the $\log |S|$ tree levels, that is, the root node, would yield $\eta_K(x)$ for all $x \in X$. Figure 4.5 provides a complete algorithm. (For simplicity, we assume that $|S| = 2^n$ for some integer $n > 0$.)

The binary-addition algorithm contains three nested loops:

1. the outermost loops over the $\log |S|$ branch levels in the binary tree;

2. the second loops over the $|S|/2^\ell$ nodes in branch level $\ell$ of the binary tree; and

3. the innermost loops over the $|X|$ master-problem solutions to combine arrays.

At each level of the tree $\ell = 1, \ldots, \log |S|$, a parallel implementation could update the arrays $\eta^m$ for all $m = 1, \ldots, |S|/2^\ell$ in parallel, on $|S|/2^\ell$ processors (Steps 3 through 5).

Note that the binary-addition method can employ no more than $|S|/2$ parallel processors because there are only $|S|/2$ pairs of cuts. If $N \geq |S|/2$, then the parallel implementation could, at least in theory, reduce objective-function calculation time (Step 8b) from $T_1$ to $T_N = T_1 \cdot \log |S|/|S|$. For $N = |S|$, this yields $E = (T_1)/(|S| \cdot T_1 \cdot \log |S|/|S|) = 1/\log |S|$. (Since the method can only employ $|S|/2$ processors, letting $N = |S|/2$ would leave the processing time unchanged while doubling the objective-function calculation’s efficiency.)

Figure 4.5: Binary-addition method for calculating $\eta(x)$. Assumes that $|S| = 2^n$ for some integer $n > 0$.

| 1: for ($\ell = 1$ to $\log |S|$) |
| 2: for ($m = 1$ to $|S|/2^\ell$) |
| 3: for (all $x \in X$) |
| 4: if ($\ell = 1$) $\{ \eta^m(x) \leftarrow p^m \eta^m(x) + p^{m+|S|/2^\ell} \eta^{m+|S|/2^\ell}(x) \}$ |
| 5: else $\{ \eta^m(x) \leftarrow \eta^m(x) + \eta^{m+|S|/2^\ell}(x) \}$ |
| 6: Report $\eta^1(x)$ as solution; Stop. |
4.6 Computational Results

This section applies SCA, MCA, SCEA, and MCEA to the stochastic capacitated facility-location problem of Sanchez and Wood (2006), in which customer demands for a single product are uncertain. This simple, parameterized model can be varied in size for algorithmic testing, yet it still resembles certain real-world facility-location models (e.g., Chan et al., 2001; Galvão et al., 2002; Vasko et al., 2003). We provide an overview of the model here; the appendix presents a complete formulation.

SCFL describes facility-construction and product-distribution decisions for a manufacturing company. In the first stage, the manufacturer decides which $b$ (or fewer) facilities to construct at $I$ potential sites, $I > b$. In the second stage, the manufacturer observes random demands from $J$ customers and then meets those demands (i.e., manufactures and ships the product) at minimum cost while paying a per-unit penalty for unserved demand. The manufacturer’s objective is to minimize facility-construction costs plus expected product-distribution costs.

For ease of exposition, we use $P(I, J, b, \beta, |S|)$ to describe the key attributes of SCFL. $I$ and $J$ are defined above, while (i) $b$ is the maximum number of facilities that may be constructed, (ii) $\beta$ is a parameter that controls the variance of the demand distributions, and (iii) $|S|$ is the number of scenarios. An instance of $P(I, J, b, \beta, |S|)$ also contains location, capacity, and cost data as well as $|S|$ scenario demands for each customer $j$, i.e., $d^j_s$. These values are drawn randomly from a uniform distribution on $[\mu_j - \beta \mu_j, \mu_j + \beta \mu_j]$, where $\mu_j$ is a parameter that represents the expected demand from customer $j$.

We implement SCA, MCA, SCEA, and MCEA in C++ using up to eight Intel Xeon E5520 quad-core processors on a computer with 12 GB of memory. We solve all scenario subproblems, SCA and MCA master problems, and monolithic MIPs on the same computer using CPLEX solver 12.4 (IBM, 2011) with default parameter settings except that we specify deterministic parallelism with either one, four, or eight threads, as described below.

Our methods employ the following parallelism:
• SCA solves master problems using parallel B&B with one, four, or eight threads.

• MCA solves master problems using parallel B&B with one, four, or eight threads.

• SCEA does not use parallel processing.

• MCEA updates subfunctions using OpenMP parallelization (OpenMP Architecture Review Board, 2014) with one, four, or eight processors.

• CPLEX solves the monolithic MIP using parallel B&B with one, four, or eight threads.

For simplicity, we refer to CPLEX’s parallel threads as “parallel processors.” Since the default relative optimality tolerance permits CPLEX to return an “optimal” objective value for \(\text{MP}(\mathcal{X}, \mathcal{A}_K)\) that exceeds \(z^*_K\) by up to 0.01%, our SCA and MCA implementations calculate \(\bar{z}\) using the CPLEX-computed “best node” value. (We find that order-of-magnitude changes in the CPLEX relative optimality tolerance have little impact on overall solution times; thus, we use the default value of \(10^{-4}\).) SCEA and MCEA each solve master problems exactly.

4.6.1 Tests on Instances of \(P(20, 50, 10, \beta, |S|)\)

We begin by investigating four groups of \(P(20, 50, 10, \beta, |S|)\) instances, one for each combination of \(\beta \in \{0.1, 0.4\}\) and \(|S| \in \{24, 48\}\); these particular values of \(|S|\) permit scenarios to be divided evenly among \(N = 2, 3, 4, 6, 8\) parallel processors. For each group, we (i) randomly generate ten model instances and solve them using \(\varepsilon_{BD} = 0.05\), and (ii) randomly generate ten new model instances and solve these using \(\varepsilon_{BD} = 0.01\). (Recall that \(\varepsilon_{BD}\) is the allowable Benders optimality gap.)

Each Benders algorithm solves each SCFL instance with \(\varepsilon_{BD}\) and a single processor. All but SCEA also solve each instance again with \(N = 4\) and \(N = 8\) processors. (Recall that, since \(|\mathcal{X}|\) is only \(6.17 \times 10^5\), we do not implement parallel SCEA.) A limit of 3,600 seconds applies on the time that each Benders method spends solving any instance. (For any particular combination of \(\varepsilon_{BD}, |S|, \beta,\) and \(N\), we abandon a solution method once it fails
Table 4.1: Statistics to compare B&B and various decomposition algorithms for solving ten random model instances of $P(20, 50, 10, \beta, |S|)$, using $N=1, 4, \text{ and } 8$ processors.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>24</td>
<td>0.1</td>
<td>1</td>
<td>1</td>
<td>1.9</td>
<td>2.3</td>
<td>1.3</td>
<td>1.8</td>
<td>-</td>
<td>6.7</td>
<td>13.6</td>
<td>16.8</td>
<td>0.5</td>
<td>0.4</td>
<td>26.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>1.6</td>
<td>1.4</td>
<td>1.7</td>
<td>23.6</td>
<td>6.4</td>
<td>12.0</td>
<td>16.8</td>
<td>-</td>
<td>-</td>
<td>4.7</td>
<td>3.0</td>
<td>18.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>1.6</td>
<td>1.3</td>
<td>2.0</td>
<td>19.6</td>
<td>6.1</td>
<td>9.9</td>
<td>16.8</td>
<td>-</td>
<td>-</td>
<td>4.5</td>
<td>3.0</td>
<td>18.1</td>
</tr>
<tr>
<td>0.4</td>
<td>26.0</td>
<td>3.3</td>
<td>1</td>
<td>1.4</td>
<td>0.7</td>
<td>1.8</td>
<td>23.5</td>
<td>5.2</td>
<td>7.9</td>
<td>16.8</td>
<td>-</td>
<td>-</td>
<td>4.5</td>
<td>2.6</td>
<td>16.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>1.4</td>
<td>0.7</td>
<td>2.3</td>
<td>25.2</td>
<td>5.4</td>
<td>7.5</td>
<td>16.8</td>
<td>-</td>
<td>-</td>
<td>4.4</td>
<td>2.5</td>
<td>16.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>1.4</td>
<td>0.7</td>
<td>2.3</td>
<td>25.2</td>
<td>5.4</td>
<td>7.5</td>
<td>16.8</td>
<td>-</td>
<td>-</td>
<td>4.4</td>
<td>2.5</td>
<td>16.9</td>
</tr>
<tr>
<td>6</td>
<td>25.8</td>
<td>3.3</td>
<td>1</td>
<td>4.6</td>
<td>1.7</td>
<td>3.3</td>
<td>32.4</td>
<td>31.9</td>
<td>42.4</td>
<td>26.0</td>
<td>-</td>
<td>-</td>
<td>13.3</td>
<td>8.3</td>
<td>24.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>4.6</td>
<td>1.7</td>
<td>3.9</td>
<td>32.3</td>
<td>30.7</td>
<td>38.5</td>
<td>26.1</td>
<td>-</td>
<td>-</td>
<td>12.5</td>
<td>8.0</td>
<td>24.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>4.6</td>
<td>1.7</td>
<td>4.0</td>
<td>32.3</td>
<td>30.7</td>
<td>38.5</td>
<td>26.1</td>
<td>-</td>
<td>-</td>
<td>12.5</td>
<td>8.0</td>
<td>24.9</td>
</tr>
<tr>
<td>48</td>
<td>25.5</td>
<td>5.8</td>
<td>1</td>
<td>2.5</td>
<td>0.7</td>
<td>3.0</td>
<td>32.5</td>
<td>25.8</td>
<td>57.8</td>
<td>19.1</td>
<td>-</td>
<td>-</td>
<td>9.6</td>
<td>5.0</td>
<td>19.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>4.6</td>
<td>1.7</td>
<td>3.3</td>
<td>32.4</td>
<td>31.9</td>
<td>42.4</td>
<td>26.0</td>
<td>-</td>
<td>-</td>
<td>13.8</td>
<td>5.2</td>
<td>35.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>4.6</td>
<td>1.7</td>
<td>3.9</td>
<td>32.3</td>
<td>30.7</td>
<td>38.5</td>
<td>26.1</td>
<td>-</td>
<td>-</td>
<td>13.8</td>
<td>5.2</td>
<td>35.9</td>
</tr>
<tr>
<td>0.05</td>
<td>5</td>
<td>1.1</td>
<td>1</td>
<td>5.2</td>
<td>4.2</td>
<td>10.2</td>
<td>27.9</td>
<td>90.0</td>
<td>341.0</td>
<td>70.1</td>
<td>2.0</td>
<td>1.1</td>
<td>95.1</td>
<td>10.8</td>
<td>94.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>4.4</td>
<td>3.4</td>
<td>21.5</td>
<td>90.2</td>
<td>120.4</td>
<td>206.5</td>
<td>69.9</td>
<td>-</td>
<td>-</td>
<td>17.9</td>
<td>9.3</td>
<td>69.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>4.5</td>
<td>3.2</td>
<td>15.1</td>
<td>90.1</td>
<td>119.1</td>
<td>186.9</td>
<td>70.3</td>
<td>-</td>
<td>-</td>
<td>15.7</td>
<td>8.8</td>
<td>69.6</td>
</tr>
<tr>
<td>0.4</td>
<td>9.2</td>
<td>5.0</td>
<td>1</td>
<td>2.4</td>
<td>2.3</td>
<td>9.2</td>
<td>89.5</td>
<td>47.3</td>
<td>29.4</td>
<td>56.5</td>
<td>-</td>
<td>-</td>
<td>14.8</td>
<td>6.0</td>
<td>55.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>2.5</td>
<td>2.3</td>
<td>10.9</td>
<td>89.5</td>
<td>47.2</td>
<td>28.5</td>
<td>56.5</td>
<td>-</td>
<td>-</td>
<td>13.0</td>
<td>4.6</td>
<td>55.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>2.5</td>
<td>2.3</td>
<td>9.6</td>
<td>89.5</td>
<td>47.2</td>
<td>28.5</td>
<td>56.5</td>
<td>-</td>
<td>-</td>
<td>13.0</td>
<td>4.6</td>
<td>55.9</td>
</tr>
<tr>
<td>48</td>
<td>22.3</td>
<td>12.5</td>
<td>1</td>
<td>12.0</td>
<td>9.8</td>
<td>37.9</td>
<td>131.4</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>38.3</td>
<td>35.7</td>
<td>80.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>12.0</td>
<td>10.0</td>
<td>37.1</td>
<td>131.4</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>33.2</td>
<td>31.0</td>
<td>80.4</td>
</tr>
</tbody>
</table>

Notes: CPLEX solves the monolithic MIP with a relative optimality tolerance of $\varepsilon_{BD}$, and solves SCA and MCA master problems using its default relative optimality tolerance of $10^{-4}$. CPLEX solves the monolithic MIP, and the SCA and MCA master problems using $N$ parallel processors. MCEA updates subfunctions using $N$ parallel processors. Bold font indicates the fastest average solution time for each group. * indicates that one or more instances do not solve to optimality in 3,600 seconds. - indicates that parallel processing was not implemented for BDEA.

To solve any instance in 3,600 seconds.) To compare Benders decomposition to branch-and-bound, CPLEX solves each instance as a monolithic MIP with relative optimality tolerance $\varepsilon_{MIP}$ ($\varepsilon_{MIP} = 0.05$ or $\varepsilon_{MIP} = 0.01$) and with one, four, and eight processors. Table 4.1 provides solution times (which do not include model-generation time) and other related statistics.

We find that for each group of problems:

- SCEA (MCEA) is faster than SCA (MCA), and yields solution times with less variability than SCA (MCA).

- SCEA (MCEA) requires approximately the same number of iterations as SCA (MCA).
• SCEA is faster than MCEA.

• SCEA is faster than solving the monolith.

• MCA is uncompetitive with other methods, and we drop it from here on.

Salmerón and Wood (2014) note that exact master-problem solutions, like those generated by SCEA and MCEA, may result in a greater number of Benders decomposition iterations than would be produced by the near-optimal solutions that B&B generates for SCA and MCA. That does not appear to be an issue here, however.

The instances in Table 4.1 yield small monolithic MIPs. Larger instances are common, however (e.g., Chen et al., 2006; Silva and Wood, 2006; Schütz et al., 2008), and perhaps are more appropriate for a decomposition method. To test the ability of the BDEA algorithms to solve SCFL model instances with a greater number of scenarios, we next investigate additional groups of problems, one defined for each combination of $\beta \in \{0.1, 0.4\}$ and $|S| \in \{128, 256, 512\}$. For each group, we evaluate ten randomly generated model instances of $P(20, 50, 10, \beta, |S|)$.

We begin by solving each SCFL instance using SCA and MCEA with $N = 1, 4, \text{ and } 8$ processors, and using SCEA with $N = 1$. For each method, $\varepsilon_{BD} = 0.01$. For comparison, CPLEX solves the monolithic MIP with a relative optimality tolerance $\varepsilon_{MIP} = 0.01$. When solving the monolith, we only consider $N = 8$ parallel processors, because CPLEX tends to perform best with between four and twelve parallel threads (E. Klotz, personal communication, March 20, 2014), and we have eight processors available. Table 4.2 provides the results.

We find that:

• SCEA is the fastest solution method, and can reduce average solution times by up to 74% compared to SCA.

• SCA and MCEA are each faster than solving the monolith.
### Table 4.2: Statistics to compare B&B and various decomposition algorithms for solving ten random model instances of $P(20, 50, 10, β, |S|)$, using $ε_{BD} = 0.01$.

| $|S|$ | β   | Soln. time (sec.) | Soln. time (sec.) | Soln. time (sec.) | Soln. time (sec.) | Soln. time (sec.) | Soln. time (sec.) |
|------|-----|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| 128  | 0.1 | 117.6 80.3        | 26.0 21.8         | 111.0 28.2        | 19.5 110.8        | 8.4 4.2           | 111.8 91.6        |
|      | 0.4 | 59.8 36.9         | 64.0 90.6         | 151.4 56.3        | 59.0 151.8        | 14.4 9.7          | 150.4 96.9        |
| 256  | 0.1 | 293.0 234.3      | 19.9 21.8         | 69.9 23.5         | 21.7 70.0         | 10.1 7.7          | 71.8 61.3         |
|      | 0.4 | 236.6 178.8      | 38.3 68.9         | 94.7 42.9         | 62.0 94.6         | 16.9 17.0         | 94.0 124.5        |
| 512  | 0.1 | 1425.9 705.6     | 121.9 269.3       | 130.6 91.4        | 128.9 130.7       | 35.9 30.9         | 130.7 298.3       |
|      | 0.4 | 942.8 727.2      | 66.1 35.7         | 131.6 88.7        | 37.6 131.4        | 46.7 18.3         | 130.2 384.6       |

**Notes:**
- CPLEX solves the monolithic MIP with a relative optimality tolerance of $ε_{MIP} = 0.01$; and solves BDA and MC-BDA master problems using its default relative optimality tolerance of $10^{-4}$.
- CPLEX solves the monolithic MIP and the SCA master problems using the number of processors shown.
- MCEA updates subfunctions using the number of processors shown.
- Bold font indicates the fastest average solution time for each group.

- MCEA requires fewer iterations than SCEA (about 40 fewer, on average), but longer master-problem solution times negate any computational advantage.
- MCEA’s solution speed improves, relative to solving the monolith, as the number of scenarios and/or the number of processors increase.
- SCA’s performance does not necessarily improve with additional processors. (That is, the 8-processor solution of SCA’s master problem can actually take longer than the single processor solution.)

### 4.6.2 Parallel Processing Gains for Single-Cut Benders Decomposition Algorithm Using Enumeration to Solve Master Problems (SCEA)

While the current chapter does not implement parallel processing for SCEA, we are interested in understanding the speedup that may be possible with a parallel implementation. Section 4.5.1, above, discusses two “key tasks” that SCEA can execute in parallel during each iteration:

1. Step 3: each of $N ≤ |S|$ processors can solve $|S|/N$ subproblems (let $|S|$ be evenly divisible by $N$); and
2. Step 8a: each of up to \( N \leq |\mathcal{X}| \) processors, at least in theory, can update a portion of \( \eta_K(x) \).

Nielsen and Zenios (1997) implement a BDA that solves up to 512 subproblems in parallel on a CM-5 massively-parallel computer; modern CM-5 computers contain as many as 1024 processors (Top, 2014). Since it is certainly possible to solve 1024 scenario subproblems in parallel, and our single-processor SCEA implementation can solve 1024-scenario instances of 2SSP relatively quickly, thus providing key-task processing times, we consider solving \( P(20, 50, 10, 0.4, 1024) \) on 1024 parallel processors.

We use the C++ system clock (which has a period of 1 microsecond on our computer) to measure the time required by SCEA to solve subproblems, to update the objective function, and to execute all remaining algorithm steps while solving ten random instances of \( P(20, 50, 10, 0.4, 1024) \) using a single processor.

To calculate an upper bound for the time required by SCEA to solve each instance of \( P(20, 50, 10, 0.4, 1024) \) on a 1024-processor computer, let \( i = 1, \ldots, 10 \) index the instances; let \( k = 1, \ldots, K_i \) index the Benders iterations for instance \( i \); and

1. let \( \bar{t}_{ik}^{\text{sub}} \) be the maximum of the \( |S| \) subproblem solution times, in Step 3, for instance \( i \) during iteration \( k \);
2. let \( \bar{t}_{ik}^{\text{obj}} \) be the time required to update the objective function in Step 8a, for instance \( i \) during iteration \( k \); and
3. let \( t_i^{\text{other}} \) be the total time required for the remaining algorithm steps (1-2, 4-7, 8b, and 9-11) for instance \( i \).

A reasonable upper bound on the SCEA solution time for each instance \( i \) is:

\[
t_i = \sum_{k=1}^{K_i} \left( \bar{t}_{ik}^{\text{sub}} + \bar{t}_{ik}^{\text{obj}} \right) + t_i^{\text{other}} \quad \text{for } i = 1, \ldots, 10, \tag{4.17}
\]
where $K_i$ is the number of SCEA iterations required for instance $i$. Using subproblem solution times, objective function update times, and the time required for other algorithm steps when solving the ten instances of $P(20, 50, 10, 0.4, 1024)$, we estimate that a 1024-processor SCEA implementation could solve a typical instance in $\frac{1}{10} \cdot \sum_{i=1}^{10} t_i = 2.27$ seconds, which represents a potential 96% reduction in solution time compared to the single-processor SCEA implementation’s average solution time of 68.3 seconds. Calculating $\eta_K(x)$ in parallel could, perhaps, reduce the solution time further.

The tests on instances of $P(20, 50, 10, 0.4, 1024)$ indicate that computing time for SCEA is distributed as follows: (i) solving subproblems 97.03%, (ii) updating $\eta_K(x)$ 1.00%, and (iii) all remaining tasks 1.97%. If we simply solve subproblems in parallel on eight processors and assume that all subproblems take the same amount of time, then we can estimate an average SCEA solution time for $P(20, 50, 10, 0.4, 128)$ as

$$0.9703 \cdot \frac{14.4}{8} + (1 - 0.9703) \cdot 14.4 = 2.2$$

seconds, where 14.4 seconds is the average time required to solve an instance of $P(20, 50, 10, 0.4, 128)$ using single-processor SCEA (Table 4.2). This would represent (i) an 85% reduction in solution time compared to the serial SCEA implementation, and (ii) a 95% reduction in solution time compared to an SCA implementation that solves subproblems in parallel on eight processors, which we estimate would require $56.3 - (14.4 - 2.2) = 44.1$ seconds on average.

### 4.6.3 Parallel Processing Gains for Multicut Benders Decomposition Algorithm Using Enumeration to Solve Master Problems (MCEA)

$\text{2SSP}(\mathcal{X}, \mathcal{Y}, \mathcal{S})$ is not a problem for which multicut decomposition is likely to prove more effective than single-cut decomposition, since the number of scenario subproblems $|\mathcal{S}|$ is much larger than the number of first-stage constraints (Birge and Louveaux, 2011, p. 200). (For $\text{2SSP}(\mathcal{X}, \mathcal{Y}, \mathcal{S})$, there is a single first-stage constraint.) Although $\text{2SSP}(\mathcal{X}, \mathcal{Y}, \mathcal{S})$ is not well
suited to multicut decomposition, other problems may be; for example, see Teng et al. (2004) and You et al. (2009). Thus, we will note the key tasks that MCEA can execute in parallel during each iteration (let $|S|$ be evenly divisible by $N$):

1. Step 3: each of $N \leq |S|$ processors can solve $|S|/N$ subproblems;

2. Step 8a: each of $N \leq |S|$ processors can update $|S|/N$ subfunctions; and

3. Step 8b: up to $N \leq |S|/2$ processors can calculate the objective function using binary addition.

4.7 Conclusion

This chapter has extended the “Benders decomposition algorithm using enumeration to solve master problems” (BDEA) to mixed-integer programs (MIPs) with multiple independent subproblems. We implement “single-cut” (SCEA) and “multicut” (MCEA) versions of BDEA and demonstrate that the methods produce substantial reductions in solution times for a stochastic capacitated facility-location problem (SCFL). We implement the algorithms and solve the master problems with C++, while CPLEX solves the scenario subproblems.

For models that are large enough for standard Benders decomposition to become attractive compared to solving the monolith, we find that a single-processor SCEA implementation can reduce solution times by up to 74% compared to an eight-processor single-cut Benders decomposition algorithm (SCA) that does not solve subproblems in parallel. Although MCEA is not as fast as SCEA, the multicut implementation does solve large instances for which a standard multicut Benders decomposition algorithm (MCA) is too slow for practical use.

We estimate that solving subproblems in parallel on eight processors could reduce SCEA solution times by up to 85%. We also determine a reasonable upper bound on the time required to solve a 1024-scenario SCFL instance using SCEA with 1024 parallel processors: this massively parallel implementation could reduce solution times by up to 96% compared to a single-processor implementation.
Previous Benders decomposition methods have been able to take advantage of parallel computing to solve independent subproblems. Parallel processing of master problems has been limited, however, to the multi-threading capabilities of commercially available branch-and-bound solvers. Using SCEA and MCEA, we can now explicitly parallelize a master-problem solution method.

We have studied a special-case MIP in which $\mathcal{X}$ is defined by a single knapsack constraint. However, we are also interested in solving more general MIPs with more complicated definitions for $\mathcal{X}$. Future work will investigate such extensions.
CHAPTER 5
GENERAL CONCLUSION

Up until now, no extension of Benders decomposition to general integer programs (IPs) existed. Furthermore, a Benders decomposition algorithm (BDA) could become too slow for practical use as its master problems became more difficult with each Benders iteration. This thesis has made the extension to IPs, and used explicit enumeration to reduce Benders master-problem solution times by an order-of-magnitude for certain problem types.

5.1 Research Contributions

Chapter 2 extends BDA to pure integer models by solving a mixed-integer programming (MIP) relaxation of an IP with BDA, using a local-search technique to identify promising pure integer solutions, applying solution-elimination constraints (SECs) to ensure convergence, and continuing BDA iterations, if needed. This extension solves the IP while maintaining a tractable master problem and avoiding complicated subproblem cuts. Our main contributions are techniques that identify promising integer solutions from a given MIP solution and that introduce SECs to ensure convergence. Testing indicates that the method solves certain set-packing instances faster than a standard branch-and-bound algorithm; an enumerative master-problem solution technique also seems likely to improve the algorithm’s solution speed.

Chapter 3 develops the “Benders decomposition algorithm using enumeration to solve master problems” (BDEA), which stores master-problem objective-function values for each feasible master-problem solution. After identifying a new Benders cut, we solve the updated master problem by enumerating the feasible solutions, and for each solution (i) calculating the lower bound (for a minimizing problem) that the new cut provides, (ii) comparing this new lower bound to the previous lower bound (that is, to the stored objective-function value), and (iii) saving the greater (that is, the more restrictive) of the two bounds as the
new objective-function value. Although the computational results in the first implementation of BDEA in the literature were inconclusive, we have shown that enumeration can reduce master-problem solution times by an order-of-magnitude compared to solving the same problems using branch-and-bound.

Chapter 4 implements “single-cut” and “multicut” versions of BDEA to solve mixed-integer programs with multiple independent subproblems. For 128-scenario stochastic capacitated facility-location problem instances, single-cut BDEA reduces average solution times by up to 74% compared to single-cut BDA. By solving subproblems in parallel, an eight-processor single-cut BDEA could reduce the average solution times by an estimated 95% compared to a single-cut BDA that also solves subproblems in parallel.

5.2 Suggested Further Research

The IBDA implementation in Chapter 2 could be improved to increase solution speeds. Opportunities include (i) implementing IBDA in a compilable language such as C++, and (ii) solving Benders decomposition master problems using explicit enumeration rather than B&B. These improvements could make IBDA competitive with B&B, even for IPs whose MIP relaxations do not necessarily yield integer solutions.

The BDEA implementations from Chapter 4 could be extended to solve two-stage stochastic MIPs for which the master-problem solution space $X$ is defined by a knapsack constraint and some set of additional constraints. This new implementation could simply enumerate all solution vectors that satisfy the knapsack constraint, and then check for feasibility with the set of additional constraints. It may be possible, in some cases, to further improve the enumeration by applying Lagrangian relaxation to the set of additional constraints rather than applying this “complete” relaxation.

Finally, the BDEA implementations could be improved by increasing the degree of parallel processing. One key opportunity for improvement would be to solve the independent subproblems in parallel. Additional opportunities include calculating master-problem objective-function values in parallel by either partitioning the feasible master-problem solutions into
subsets, or by summing the “subfunctions” in parallel using “binary addition.”
REFERENCES CITED


APPENDIX - STOCHASTIC CAPACITATED FACILITY-LOCATION PROBLEM

This appendix describes the stochastic capacitated facility-location problem SCFL with random customer demands $\tilde{d}$ for a single product, and a finite number of demand scenarios $|S|$, each with probability $p^s$ (Sanchez and Wood, 2006). Table A.1 presents indices, parameters, and variables for SCFL.

The problem formulation is:

$$\text{SCFL} \quad z^* = \min_{x \in X} \sum_{i \in I} c_i x_i + \sum_{s \in S} p^s z^s(x), \text{ where for all } s \in S,$$

$$z^s(x) = \min_{y \geq 0, v \geq 0} \sum_{i \in I} \sum_{j \in J} f_{ij} y_{ij} + \sum_{j \in J} r_j v_j,$$

s.t. \quad \sum_{j \in J} y_{ij} \leq u_i x_i \forall i \in I \quad \text{(A.3)}

$$\sum_{i \in I} y_{ij} + v_j = d^s_j \forall j \in J,$$

\text{(A.4)}

Table A.1:  Stochastic capacitated facility-location description from Sanchez and Wood (2006).

<table>
<thead>
<tr>
<th>Indices of Sets</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i \in I$</td>
<td>candidate facility locations</td>
</tr>
<tr>
<td>$j \in J$</td>
<td>customer locations</td>
</tr>
<tr>
<td>$s \in S$</td>
<td>demand scenarios</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_i$</td>
<td>fixed cost to construct facility $i$ ($$$)</td>
</tr>
<tr>
<td>$b$</td>
<td>maximum number of facilities that may be constructed</td>
</tr>
<tr>
<td>$u_i$</td>
<td>capacity of facility $i$ (tons), if constructed</td>
</tr>
<tr>
<td>$f_{ij}$</td>
<td>shipping cost from facility location $i$ to customer $j$ ($$/ton)</td>
</tr>
<tr>
<td></td>
<td>(Euclidean distance between facility location $i$ and customer $j$)</td>
</tr>
<tr>
<td>$\tilde{d}_j$</td>
<td>random demand for product at customer $j$ (tons)</td>
</tr>
<tr>
<td>$r_j$</td>
<td>penalty for unmet demand at customer $j$ ($$/ton)</td>
</tr>
<tr>
<td>$p^s$</td>
<td>probability that scenario $s$ occurs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$</td>
<td>1 if facility $i$ is constructed, 0 otherwise</td>
</tr>
<tr>
<td>$y_{ij}$</td>
<td>product shipped from facility $i$ to customer $j$ (tons)</td>
</tr>
<tr>
<td>$v_j$</td>
<td>unmet demand for product at customer $j$ (tons)</td>
</tr>
</tbody>
</table>
and where \( \mathcal{X} \equiv \{ \mathbf{x} \in \{0,1\}^{|\mathcal{I}|} | \sum_{i \in \mathcal{I}} x_i \leq b \} \), and \( p^* = 1/|\mathcal{S}| \) for all \( s \in \mathcal{S} \). Note that \( \sum_{s \in \mathcal{S}} p^s z^s(\hat{x}) \) represents the expected value of the company’s shipping costs plus unmet demand costs, given facility-construction plan \( \hat{x} \). The model’s only stochastic elements are the demands \( d^s_j \), which are drawn randomly from a uniform distribution on \( [\mu_j - \beta \mu_j, \mu_j + \beta \mu_j] \), where \( \mu_j \) is a deterministic expected demand for customer \( j \) and \( \beta \in [0, 1] \) is a parameter. (If \( \beta \equiv 0 \), SCFL becomes a deterministic capacitated facility-location problem as in Wentges, 1996.)

Once the user selects a facility-construction plan and a demand vector \( \mathbf{d}^s \) is realized, SCFL yields the following scenario subproblem:

\[
\text{SUB}^s(\hat{x}) \quad z^s(\hat{x}) = \min_{y \geq 0, v \geq 0} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} f_{ij} y_{ij} + \sum_{j \in \mathcal{J}} r_j v_j \quad \text{subject to} \quad \sum_{j \in \mathcal{J}} y_{ij} \leq u_i \hat{x}_i \quad \forall i \in \mathcal{I} \quad [\alpha^s(\hat{x})] \quad (A.5)
\]

\[
\sum_{i \in \mathcal{I}} y_{ij} + v_j = d^s_j \quad \forall j \in \mathcal{J} \quad \gamma^s(\hat{x}) \quad (A.6)
\]

where (i) \( y_{ij} \) is the quantity, in tons, shipped from facility \( i \) to customer \( j \); (ii) \( v_j \) is the unserved demand, in tons, for customer \( j \); and (iii) the optimal dual variables are shown in brackets. Define \( \hat{\alpha}^s \equiv \alpha^s(\hat{x}) \) and \( \hat{\gamma}^s \equiv \gamma^s(\hat{x}) \). Since a scenario subproblem is a transportation problem with elastic demands, it is feasible for any \( \mathbf{x} \in \mathcal{X} \). Therefore, SCFL exhibits relatively complete recourse (Wets, 1974).

Benders decomposition reformulates SCFL into an equivalent multicut master problem:

\[
\text{MC-MP}[\mathcal{A}^s, \mathcal{G}^s] \quad z^*(\mathcal{A}, \mathcal{G}) = \min_{\mathbf{x} \in \mathcal{X}} \sum_{i \in \mathcal{I}} c_i x_i + \sum_{s \in \mathcal{S}} p^s \eta^s \quad \text{subject to} \quad \eta^s \geq \sum_{i \in \mathcal{I}} \hat{\alpha}_i^s u_i x_i + \sum_{j \in \mathcal{J}} \hat{\gamma}_j^s d^s_j \quad \forall s \in \mathcal{S}, \quad (\hat{\alpha}^s, \hat{\gamma}^s) \in (\mathcal{A}^s, \mathcal{G}^s), \quad (A.9)
\]

where \((\mathcal{A}^s, \mathcal{G}^s)\) is the set of all dual (extreme-point) vectors for \( \text{SUB}^s(\hat{x}) \). We omit the standard master problem formulation, which follows directly.
Table A.2: Parameters for stochastic, capacitated facility-location test problems.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_j = r$</td>
<td>$2 \cdot \max_{(i,j) \in \mathcal{I}, j \in \mathcal{J}} f_{ij} \forall j \in \mathcal{J}$</td>
</tr>
<tr>
<td>$\mu_j$</td>
<td>$U[50, 150] \forall j \in \mathcal{J}$</td>
</tr>
<tr>
<td>$u_i$</td>
<td>$\left[U[0.5, 1.5] \left(\sum_{j \in \mathcal{J}} \mu_j\right) / b\right] \forall i \in \mathcal{I}$</td>
</tr>
<tr>
<td>$c_i$</td>
<td>$u_i r / b \forall i \in \mathcal{I}$</td>
</tr>
</tbody>
</table>

Sanchez and Wood (2006) provide values for $|\mathcal{I}|$, $|\mathcal{J}|$, $b$, and $\beta$ and also note that the facilities are assigned randomly, a priori, within a rectangle that is one unit high by three units wide. These authors have also provided definitions for the remaining problem parameters, which are not noted in Sanchez and Wood (2006). Table A.2 contains these definitions obtained from the authors.